ONRL Report

8-010-C

European Seminar on Neural Computing

Claire Zomzely-Neurath

31 August 1988

U.S. Office of Naval Research, London

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS ~~AD-A209 444~~
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) 8-010-C	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Office of Naval Research Branch Office, London	6b OFFICE SYMBOL (If applicable) ONRBRO
6c ADDRESS (City, State, and ZIP Code) Box 39 FPO, NY 09510-0700	7a NAME OF MONITORING ORGANIZATION 7b ADDRESS (City, State, and ZIP Code)
8a NAME OF FUNDING / SPONSORING ORGANIZATION	8b OFFICE SYMBOL (If applicable)
8c ADDRESS (City, State, and ZIP Code)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER 10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)

European Seminar on Neural Computing

12 PERSONAL AUTHOR(S)

Claire Zomzely-Neurath

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Conference	FROM _____ TO _____	31 August 1988	35

16 SUPPLEMENTARY NOTATION

17 COSATI CODES		
FIELD	GROUP	SUB-GROUP
12	05	12 09
12	06	

18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

neural computing; connectionist models; programing languages; parallel architectures; Boltzmann machine; parallel neural computers

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The presentations given at this seminar, held in February 1988 in London, UK are reviewed in depth. Topics range from neural systems and models through languages and architectures to the respective European and American perspectives on neurocomputing.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED

22a NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE (Include Area Code)	22c OFFICE SYMBOL
C.J. Fox	(44-1) 409-4340	310

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

☆ U.S. Government Printing Office: 1986—607-044

UNCLASSIFIED

Contents

EUROPEAN SEMINAR ON NEURAL COMPUTING

Introduction

This informative and focused conference dealing with a rapidly developing area was held in London, UK, over a 2-day period, February 8 and 9, 1988. The meeting was organized by IBC Technical Services Ltd., London, UK, with the scientific program arranged by leading UK scientists in the area of neural computing.

There were 122 delegates to this conference, the majority from the UK. However, nine West European countries were also represented as well as the US and Japan. Sixty-five percent of the attendees were from industrial organizations with the balance from academia.

Neural computing (based on the computational structure of the brain) is a revolutionary area providing major breakthroughs in commercial pattern recognition and learning applications. The primary application areas are: image processing, speech recognition and synthesis, control of robot motion, and authentication systems.

An international group of expert speakers presented this 2-day seminar on the application areas, the connectionist models underlying the applications, and the hardware and software systems of neural computing. The strengths, weaknesses and appropriate domain of all aspects of this technology were examined in detail.

Neural computing technologies subdivide into connectionist models and parallel neural computers. Connectionist models can be further classified into two groups: associative memory systems and learning systems, each of which was discussed. Massively parallel neural computers are required to achieve the full potential of connectionist models. These computers execute neural models much in the same way that traditional computers perform number crunching tasks. During this meeting, a review of the hardware and software requirements for neural computers was presented. In addition, three experts reviewed neural computing developments which are being commercialized in the US, Europe and Japan along with the progress being made in their respective markets.

The format, and titles of the sessions, of the scientific program was as follows:

- Neural Systems and Models
- Connectionist Models: Background and Emergent Properties
- Programing Languages for Neural Computers
- Associative Memories and Representation of Knowledge as Internal States in Distributed Systems
- Neurocomputing Applications – A United States Perspective
- Parallel Architectures for Neurocomputers
- Combinatorial Optimization on a Boltzmann Machine
- Neural Networks: A European Perspective
- Adaptation in Open Systems: Learning and Evolution
- Neurocomputing: Neurons as Microcomputers
- Trends in Neural Computing

Detailed accounts of most of the above presentations are given in the following report.

Neural Systems and Models

Neural systems and models was discussed by D. Willshaw (Center for Cognitive Science, University of Edinburgh, UK), who said that artificial neural computing systems, based on the properties of real nervous systems, have a radically different structure and mode of action from conventional computers. The prime attraction of these artificial systems, he said, lies in the hope that they may share some of the remarkable processing capabilities of real brains, by far the most sophisticated computing devices in existence.

Artificial neural networks (called neural networks) are highly abstract models of real nervous systems. Since at least the 1930's, a close relationship has existed between people interested in the design of artificial brains and those interested in the brain itself. Among other fields, those of mathematical biophysics, cybernetics, and most recently, connectionism and parallel distributed processing have embraced both points of view, which have profited from advances in related fields such as symbolic logic, automata theory, information theory and holography.

Willshaw said that it was in the late 19th century that the neuron was first established as the basic unit of the nervous system. Ever since then, the idea has been developed that the synapse – the structure through which individual nerve cells intercommunicate – is the site of learning and memory. One of the best known proposals is due to Hebb, who suggested that the pattern of neural activity is the physiological basis of learning; and that the condition for strengthening of an individual synapse is the simultaneous activity in the presynaptic and postsynaptic cells.

The most seductive feature of a neural network for many people, Willshaw said, is that it may be able to learn – and to learn by experience rather than by explicit

---

Dr. Zomzely-Neurath is the Liaison Scientist for Biochemistry, Neurosciences, and Molecular Biology in Europe and the Middle East for the Office of Naval Research's London Branch Office. She is on leave until July 1989 from her position as Director of Research, the Queen's Medical Center, Honolulu, Hawaii, and Professor of Biochemistry, University of Hawaii School of Medicine

programing. Most of the learning machines that have been proposed employ schemes similar in principle to that proposed by Hebb: that is, that the memory resides in the connections between the basic elements (hence, "connectionism"), and the strengths of these connections are changed according to the local conditions. A machine based on this principle called the Perceptron, which was designed to learn binary pattern discriminations by experience, was discussed later in the conference. Willshaw described the archetypical neural network, of which the Perceptron is a particular example.

Neural Networks

A neural network is a highly interconnected set of simple processing units which interact by means of the signals passing between. At any moment of time each unit is at a certain level of activity which is determined from the levels of activity of the units which transmit signals to it, weighted by the strengths of the appropriate interconnections. In general terms, the task of the network is to learn a mapping from inputs to outputs; that is, each possible input, represented as a pattern of activity over the units designated as input units, must elicit a certain pattern of activity over the designated output units. Learning the required mapping involves gradually changing the weights of the interconnections, and in most cases, this is an iterative procedure involving error-correction. Each input is presented in turn; the resulting output is compared with the desired output and the weights are altered in order to make the correct output more likely to occur on subsequent presentation of this input. Much of the recent interest in programing, according to Willshaw, has been due to the development of novel learning procedures.

He said that from both computational and biological points of view, systems of this type have their advantages. From the information processing perspective, the attraction is that the relationships between entities are stored directly. Associative memory is represented directly and in parallel in the hardware. The distributed nature of the representation implies that memories should be resistant to local damage. Furthermore, the system would be able to function with incomplete or inaccurate inputs, yielding the properties of content-addressability and generalization. From the neurophysiological angle, distributed associative memories have always held out the promise of a solution to the problems of equipotentiality emphasized by Lashley and to that of stimulus generalization (Hebb).

Neural Networks and the Brain

Willshaw said that three "brain-like" properties of neural networks can be identified: (1) the individual units (the nerve cells) are simple processing units, (2) learning involves the modification of strengths of the interconnections according to locally available information, and (3) the units are highly, if not completely, interconnected, giving a highly parallel architecture.

Properties of the Nerve Cell

Until very recently, the idea that synapses are modified according to the local conditions has been based on hope rather than fact. It has been extremely difficult to obtain direct evidence. Recent work, however, has indicated that a learning rule similar to that proposed by Hebb may operate. The phenomenon of Long Term Potentiation (LTP) is a long-lasting alteration in the efficacy of synaptic transmission, which was first discovered in the mammalian hippocampus. LTP results from rapid stimulation of the incoming nerve fibers. In most cases, a few impulses delivered at near-maximal rate is most effective. However, if the postsynaptic membrane is simultaneously artificially depolarized, single stimuli are sufficient. Here, according to Willshaw, a neo-Hebbian rule seems to be operating – the condition for synaptic modification is presynaptic activity together with postsynaptic depolarization, rather than postsynaptic activity as proposed by Hebb.

Neural Connectivity

The area of nervous system research from which neural networkers may learn most concerns the nature of neural connectivity. In neural networks the units are generally arranged in completely connected small groups, each of which is interconnected to give a number of layers fairly arbitrarily arranged. Much of the wiring (for example, the wiring required for execution of the back-propagation learning algorithm) is not given explicitly, according to Willshaw. In the brain there are many different types of wiring patterns. In most regions of the brain the nerve cells are not completely interconnected and the neural wiring patterns vary significantly from region to region. The connections in many sensory systems are so arranged as to produce topographic projections. In all vertebrates, for example, there is a map of the eye through the optic pathway on to the primary visual center of the brain. In the hippocampus the cells are arranged in specific layers and project onto one another and back onto themselves in a stereotyped – but as yet poorly understood – fashion. In mammals, perhaps the most exquisite neuronal circuitry is that found in the cerebellum.

The cerebellum, situated behind the cerebral cortices, is thought to be involved in the learning of the control of movement. It contains five major types of nerve cells. It receives two distinct types of input – through the mossy fibers and through the climbing fibers. The mossy fibers contact the granule cells of the cerebellum, whose axons, the parallel fibers, make contact with the Purkinje cells, the single class of output cells. There may be as many as 100,000 contacts to each Purkinje cell. The climbing fibers contact the Purkinje cells directly. According to the theories of Marr and Albus, the climbing fibers relay the direct instructions to the Purkinje cells to carry out elemental movements; the mossy fibers relay the "context" in which this movement is carried out. Willshaw said that it has been proposed that modifications of the

parallel fiber/Purkinje cell synapses according to the Hebb rule makes possible the learning of the associations between each context with the appropriate instruction. The effect is that subsequent presentation of each context is sufficient to evoke the correct Purkinje cell response. In this way, sequences of movements can be learned to be carried out automatically. Willshaw then presented slides and discussed a real neural network, the cerebellum, in detail and the role assigned to each type of cell according to the Marr-Albus theory.

Connectionist Models: Background and Emergent Properties

A talk which provided an introduction to current connectionist research areas (examined in detail in later talks at this conference) was given by M. Recce (Department of Computer Science, University College, London, UK). He said that the goal of neural computers is based on an attempt to mimic the brain's function by emulating its structure. However, it is the abstraction of these neuroscience concepts into the field of connectionism which has provided the progress in designing and programing neural computers.

The framework was established in the 1960's – prior to the current wave of interest in connectionism – by the development of perceptron models. The primary contribution during this period was a simple perceptron learning paradigm. Recce discussed this paradigm and its limitations during his presentation.

Current research in connectionism, he said, subdivides into two areas, namely, associative memories and learning systems. With associative memories, information can be retrieved based on the content of the memory (auto-associator) or a relationship between remembered pieces of information (pair-associator). With learning systems, data is presented according to a set of rules, and the task is for the system to extract the underlying patterns.

Recce said that the idea that increased understanding of the brain would lead to the design of better computer systems is not new. The theory of computation, and the first electronic computers were developed during the same period that the neuronal model of brain function was being accepted. The current model of computation was not in place, and better, smarter architectures for "electronic brains" were under investigation. The early research into how to build these computers, which today might be called main-stream artificial intelligence, was principally through examining network models of simple neurons.

Recce said that during the late 1960's the main stream of artificial intelligence research moved away from the neural network approach towards a top-down modeling of human mental processes. The motivation was that the brain is just one way of making a thinking machine, and not the optimal way for electronic technology. At the present time, according to Recce, we are experiencing a significant move back to the neural network approach. In part this is due to the increased understanding of brain function, and in part due to the increased availability of computing power for the evaluation of models. The increased level of interest is largely due to the recent success that this level has experienced and the approaching horizon of applications (see Sejnowski and Rosenberg, 1986; Rumelhart et al., 1986; Hopfield, 1982; Hopfield, 1986; and Grossberg, 1986).

Another fact stimulating research in this area, Recce said, is disappointment with the performance of the current computer technology. Current computers are having a difficult time attempting to solve pattern recognition and learning problems. Even when computers come somewhat close to solving any of these problems, the process generally requires an enormous number of computational steps.

The top-down approach of main-stream artificial intelligence and the bottom-up approach of neural networks also differ in the nearest term goals. One of the problems not immediately addressed by artificial intelligence research then is low-level pattern recognition and distributed decision making. These are both near-term goals of the neural network approach. In his talk, Recce discussed the overall characteristics of neural network models as well as the range of abilities which the networks are trying to capture. He spoke about three models with increasing complexity.

Historical Perspective

Recce said that the field of neural networks really got started in 1943 when McCulloch and Pitts (1943) showed that networks of neuronlike elements were general computing devices. That is to say that a suitably constructed network could implement any computational algorithm. The largest missing piece was the method of training, or storing information in these devices. However, Hebb (1949) described a biologically plausible means for learning. He suggested that synapses were the site of changes during adaptation, and that frequently active connections should have increased chances for being active again. This is accomplished by increasing the synaptic strength or weight. This learning rule is still the basis of many connectionist models.

In 1957 Rosenblatt (1962) invented a simple class of learning network which he called the "perceptron." Perceptrons are composed of a special class of McCulloch and Pitts neuron which is called a threshold logic unit (TLU). The TLU has a number of inputs, each associated with a weight that plays a role analogous to the synaptic strength of inputs to a neuron. The total input to the TLU is an n-dimensional vector, a pattern of activity on its individual input lines. Each component of the input vector is multiplied by the weight associated with that input line and all of these products are summed. The unit gives an output of "1" if this sum exceeds its threshold.

Otherwise it gives an output of "0". The interest in Rosenblatt's perceptron model arose, Recce said, from the result that a single layer of these units could perform pattern recognition. Rosenblatt's perceptron convergence theorem proscribed a procedure and proved that the perceptron would converge to the answer, if a suitable set of weights exists. The problem was that the perceptron would only reach the answer if the network was capable by design of solving the problem. Unfortunately, with a single layer model an appropriate set of weights does not exist for interesting problems.

Perceptron Convergence Procedure

Recce described the procedure by which this simple network can learn. Figure 1 shows a simple black box which has four inputs and two outputs. Each of the inputs is connected to each of the outputs. The weights are all initially set to zero. The procedure has two phases which are repeated several times. During the first phase the inputs and outputs are held at desired values, and a small amount, δ, is added to all weights which have both active inputs and outputs. During the second phase the inputs are clamped at the same values and the TLUS. At this point the same amount of δ is subtracted from all weights which have active outputs and inputs. If a weight was correct before this two-phase operation then its weight is unchanged, as it was incremented during phase 1 and decremented during phase 2. In all other cases, the value would have changed – and in the correct direction – as a result of this pass through both phases. Recce said that there were several examples of pattern recognition learning which this single layer model was found to accomplish.

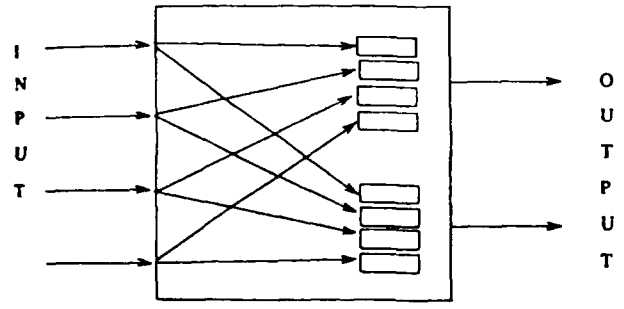


Figure 1. Simple single layer perceptron.

Throughout the late 1950's and 1960's, he said, there was a significant amount of research done on single-layer perceptrons, until a theoretical analysis of their limitations was published by Minsky and Papert (1969). These investigators set out to establish the capabilities of perceptrons, and found significant limitations. In particular, they found that single-layer perceptrons were generally less efficient at solving even the problems which they could solve than traditional computers. In particular, they found that with slightly harder problems, the perceptron required an absurdly large number of units. Two examples are parity, or the number of active bits, and connectedness, which measures if objects in the scene are connected. In addition, they addressed themselves to the potential extensions of the models to multiple layers and backward coupling. They indicated that they did not expect a dramatic increase in the abilities of perceptrons if extended in these directions, and that these extensions significantly increased the problem of treating systems analytically.

Recce said that the book published by Minsley and Papert called *Perceptrons* dampened interest in the field. The attempts to solve this set of problems was then shifted towards the now mainstream artificial intelligence research, except for a small number of researchers who remained active in this field.

Neural Network Characteristics

Neural network models, Recce said, are generally large, regular structural arrays of computationally simple, but nonlinear, processing elements which are interconnected. The research in this field has a very experimental nature, due to the mathematical complexity of these parallel nonlinear systems. In general the approach taken is to obtain the most guidance possible from analytical methods and then to conduct simulation experiments with software on serial computers or with special-purpose hardware. Therefore the richness of variation found in real neuronal networks is reduced to the simplest models which will demonstrate the behavior desired. Even with these simplifications the parameter space containing possible models is enormous.

Another characteristic of neural networks which both increases their complexity and is the key to their computational power is the distributed nature of the information stored in them. If a new piece of information is added to the network, a small change is made over the entire network rather than a single change in one place. According to Recce, this characteristic is usually a result of the rules governing the system, and also gives the system its robustness or fault tolerance.

He said that neural networks are specified by node characteristics, net topology, and training or learning rules. Within the node definition is a specification of the variability of the weights. At one extreme some models restrict these weights to a single bit (Wilshaw, 1981; Gardner-Medwin, 1976), while others use real-valued numbers (Rumelhart et al., 1986). Secondly, the response function of the node can be varied. It is usually the step function from 0 to 1 shown by (a) in Figure 2, but some models more closely model the true neuronal response (Hopfield, 1986, and b in Figure 2). Additionally the update timing of the node varies between models. In most models synchronous updating of nodes is performed, while asynchronous updating is the rule employed by the central nervous system.

The network topology varies significantly between neural network models. In the earlier perceptron models, Recce said, a single layer of units was used. This has

been extended to multilayer feed-forward networks in which the connections are just between layers of units and in which the information or analysis flows in one direction, and the correction to the performance flows in the reverse direction. Recce said that all-to-all connected networks and randomly connected networks are also frequently studied.

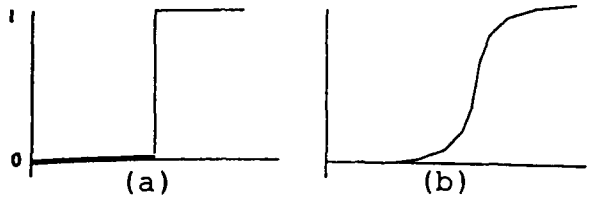


Figure 2. (a) Step function response, (b) Sigmoid response.

The largest variation in models is in the training or learning rules. These rules specify an initial set of weights and indicate how weights should be adapted during use to improve performance. Recce said that it is important to stress that only the weights change in the great majority of these models. The Hebb rule mentioned above is used by several current models and in a modified form is used by the Hopfield model (Hopfield, 1982), which was discussed in a later talk at this meeting. Rosenblatt's learning rule has been extended in various ways for the multilayer networks. One extension, called the Boltzmann Machine (Ackley et al., 1985), was discussed in detail in another talk at this meeting. Besides the required components of neural network models, many models include other characteristics, according to Recce. An example of this in the Hopfield model is symmetric interaction between units. This means that if a unit has a weight of "w" to another unit, then the other unit must have a reciprocal connection with identical weight. In another example (Rumelhart et al., 1985) with a multilayer network, units within a layer have negative weights coupling them, but have positive weights to the next layer.

Taxonomy of Goals

The tasks set for neural network models span several fields, but the underlying capabilities sought in these models fall roughly into four categories: auto-associators which relate stored patterns to themselves; pair-associators which relate patterns to a coupled pattern; supervised learning in which there is repeated correction by a teacher of errors made in the system; and unsupervised learning in which the system itself must cluster the input patterns.

In order to clarify these categories, Recce considered systems working with the letters "A" and "B". With an auto-associating system which has stored these two letters, we should expect to be able to present either letter, and it should be recognized and the states should not change. In addition, if we present a corrupted or partial letter that the system had learned we would expect the system to respond with the completed correct letter. This type of network is also called a content addressable memory.

A pair associator which had learned to associate the letter "A" with "B", should present "B" on the output if "A" is presented on the input. Similarly, if a corrupted or partial "A" is presented, then a complete "B" should appear. According to Recce, a pair-associator is significantly more valuable a device because knowledge can be represented by objects with weighted associations. Also, a pair-associator with feedback of output back to input can easily be used to construct a finite-state machine.

A supervised learning system, after repeated presentation of "A" and"B", would be able to change the representation of the input data. It could, for example, select one output line after training. The single-layer perceptron model discussed above is an example of supervised learning. A key point here, is that the correct answer is used to improve performance.

A network in the last category of unsupervised learning would be able, after successive presentation of input letters, to change the representation of the input to the output. The difference is that it performs this task without the benefit of presentation of the correct answer to the output units.

The Hopfield Model

The model initially put forward by Hopfield in 1982 is an auto-associative memory device. It has multibit weights, and units with a step function response from minus one to plus one. This response, which is thresholded at zero, is calculated by all of the units simultaneously. The network is 100 percent connected, and weights between units are symmetric. The storage rule used by the Hopfield model is a modified Hebb rule. It states that the weights should be increased by one if either both units are active (outputting a 1) or both units are inactive (outputting a -1). In the cases of one active and the other inactive the weight is decremented by 1. The network is initialized with all weights set to zero.

After the patterns have been stored in the network the recall procedure has two steps. First, an initial test pattern is presented to the network and all of the units are held momentarily at this setting. Then each unit calculates the weighted sum of its inputs, and if this is positive, outputs a 1; if this is negative, it outputs a -1. Next, each unit recalulates the weighted sum with the new set of active inputs, and recalculates its output. After this cycle is repeated a few times, the states stabilize and the output is read as the state of the units.

According to Recce, this can perform well as an auto-associator if a few conditions are met. The first condition is that the patterns stored must be different by a sufficiently large number of bits. This number is called the Hamming distance between the two patterns. The second condition is that no more than a maximum of 0.15 times N patterns can be stored in the network, where N is the number of units.

The Boltzmann Machine

This is an example of a supervised learning network. This algorithm, Recce said, is one of a few algorithms which have extended the single-layer perceptron learning algorithm to multiple layers. The reason for presenting the Boltzmann Machine rather than the more powerful "back-propogation of errors" algorithm is that this algorithm is a simple extension of the Hopfield model.

The supervised learning network is constructed in multiple layers such that there is one set of units in contact with input information, one set of units in contact with output information, and a third set of units which make no outside contacts. This third set of units have been called "hidden units." With this model, rather than explicitly storing information as would occur with an auto-associator, input patterns are presented and outputs are trained to respond correctly.

The training process has two phases, much like the phases of the perceptron learning algorithm, but with one significant change. This change is designed to allow training of the hidden units. If the network is left after presentation of the input values, it will settle into a minimum just as if it were being used to recall stored data. The problem is, according to Recce, that this might not be the global minimum. In order to convince the system to settle into the global minimum a technique called "simulated annealing" is used. The basic principle of this technique, taken from statistical mechanics, is that if a sufficient random element is added to each unit's choice of a state, then the system can escape local minima. In fact, if this randomization is allowed to persist for long enough, the system will reach a state of equilibrium. Within this equilibrium state the system will occupy minima in proportion to their size, so it will spend more time in the global minimum. Therefore, the simulated annealing technique requires setting a degree of randomization, often called the temperature, and a procedure for lowering this temperature parameter gradually to zero. In the Boltzmann Machine two phases incorporate this randomization.

During phase one, the inputs and outputs are held at the desired value – a collection of 1's and 0's. The system is brought to the equilibrium state at a low non-zero temperature. For the next short period of time the weights are modified in the following way. For each unit of time which two units are both active, the weight between them is incremented by δ. During the second phase the inputs are clamped, and the outputs are calculated. The same inputs are used. The system is again brought to equilibrium and for the same short time as before, but now decrement by δ the weights between active units. If the system was in an equilibrium state and the time was sufficiently long then the net result is that the weight should not have changed. A proof of this can be found in the paper by Ackley et. al. (1985) on Boltzmann Machines. Recce said that the results from this learning procedure are generally quite good. Many of the problems which the single-layer perceptron could not solve can be solved by this method. On the other hand, the number of units – and the configuration – required to achieve a solution is not well understood, and the time required for testing each system is significant.

According to Recce, neural network models are designed to capture aspects of pattern recgnition, learning and fault tolerance inherent in real networks of nerve cells. These goals are far from being reached by current mainstream artificial intelligence research. Neural networks seem a promising alternative, especially in light of of the current progress in this field, VLSI technology, and neuroscience. Recce said that in balance, the success seen so far should be tempered with the fact that the true understanding of the behavior of these models is quite far off, but successful application of this technology to real problems has already begun.

Programing Languages for Neurocomputers

Programing languages for neural computers still remains one of the least developed research areas. Historically, each novel class of parallel computers is associated with a corresponding class of high-level programing language. Neural computer programing languages will develop partially through absorption of appropriate concepts from current parallel languages. In his presentation, P. Treleaven (Department of Computer Science, University College, London, UK) reviewed the major classes of parallel programing languages and discussed potential contributions to neural networks. Language classes presented include Communicating Processes, Object-Oriented, Data Flow, Logic, and Semantic Network. Treleaven also discussed current proposals for neural network languages such as ANNE, CONE, P3 and NIL.

Programing Neural Networks

Over the past 2 years, interest in neural networks has surged (Hopfield, 1986; Miller, 1987; Kovaly, 1987). The reason is that neurocomputing seems to represent a fundamentally new domain of computation, complementary to traditional computing. Neural networks are massively parallel networks of primitive processing elements, based on highly simplified models of the human nervous system, exhibiting abilities such as learning, generalization and abstraction. Treleaven said that traditional computation and neurocomputation may be differentiated as: Symbol Processing – computation is specified as an explicit series of commands that manipulate symbols whether data or code; Pattern Processing – computation is specified as a network of primitive neuronlike processing elements that operate as a dynamical system through adjusting their interconnections.

Considerable research has been devoted to the design of a new generation of general-purpose parallel com-

puters and its associated programing languages – the so-called fifth-generation computer for use in the 1990's. Treleaven said that each proposed design consists of a parallel model of computation, plus a computer architecture and corresponding high-level languages, both embodying the computational model. Although these models are superficially different, all are based on programs specified as an explicit series of commands and thus are classified as performing symbol processing.

According to Treleaven, historically each new class of parallel computers is associated with a corresponding class of high-level programing languages. If the true potential of the recent advances of neurocomputing and neural networks is to be realized, then this new domain of parallel computers will require a corresponding class of general-purpose, high-level (neural network) programing languages. Neural computer programing languages, according to Treleaven, will develop through absorption of appropriate concepts from current parallel languages.

Treleaven divided his talk into four main parts: (1) a clarification of what is meant by "connectionist networks" and their relationship to semantic and neural networks; (2) a review of the major classes of parallel programing languages such as Procedural, Object-Oriented, Functional and Logic; (3) an overview of Semantic Network languages, which are closely related to neural networks; and (4) current proposals for neural network languages.

Connectionist Models

Treleaven defined the terms "connectionism," "semantic networks," and "neural networks" because, he said, there was confusion in the literature regarding these terms.

"Connectionism" refers to a broad class of massively parallel sytems composed of a large number of simple, autonomous, and interconnected processing elements. Information is principally stored in terms of the interconnection pattern among elements, and processed by means of altering these patterns or modifying the quantity known as the weight associated with each connection. At one end of the connectionist spectrum is the class of computational models known as semantic networks and at the other is neural networks.

"Semantic Networks" are basically directed graphs in which the nodes represent independent entities (i.e., concepts or objects) and the directional links represent the relationships between the entities. Computation in semantic networks usually involves creating new nodes and changing the interconnections between nodes.

"Neural Networks" are basically fully interconnected networks of nodes where computation occurs principally via dynamic change of connection strengths, modeled by the weights associated with the links.

Programing Language Properties. Treleaven said that when designing a high-level language for programing neural computers the first question to be addressed is the scope of the computational models to be embodied; that is, whether the programing language is for: (1) neural networks, (2) semantic networks, or (3) connectionism – spanning both neural and semantic networks. For programing neural networks, the main capabilities required of a language are to:

- Allow the definition of the functionality (execution instructions) for each node
- Allow the definition of the inputs and outputs of each node
- Allow the association of weights with the inputs
- Allow the linking of the inputs and outputs of nodes to form a network
- Provide the capability for controlling the relative timing in the functioning of the nodes – for example, synchronous, asynchronous, specified patterns, etc.
- Support interaction with the environment.

For programing semantic networks the main capabilities required of a language are to allow:

- Definition of the functionality of each (or a group of) nodes
- Definition of inputs and outputs of each (or a group of) nodes
- The linking (of the inputs and outputs) of nodes to form a network
- Properties to be associated with links (for example, consistent or inconsistent)
- Dynamic creation and deletion of nodes
- Dynamic creation and deletion of links.

Thus, according to Treleaven, for programing the spectrum of connectionist models, a programing language must form a superset of the mechanisms required by neural network and semantic network languages.

High-Level Programing Languages. Table 1 illustrates the various categories of programing languages and example languages. Treleaven said that in the future any of these categories of languages may become mainstream programing styles, especially when novel parallel computers sympathetic to their support become available for use. In addition, several of them could contribute to neural network languages. Treleaven then reviewed the various categories of programing languages, starting with a discussion of the most dominant, procedural programing.

Procedural Programing. This programing is based on concepts which are almost taken for granted: a global memory of cells, assignment as the basic action, and implicitly sequential control structures for the execution of statements. There are two subclasses of procedural languages, namely, the conventional sequential languages (for example, FORTRAN and BASIC) and concurrent languages (for example, ADA and OCCAM) that have parallel control structures.

Table 1. Categories of Programing Styles and Computational Models.

Category	Examples
Procedural Programming	
conventional	BASIC, FORTRAN, PASCAL
concurrent	
shared memory	ADA, MODULA-2
message passing	CSP, OCCAM
Object-Oriented Programming	SMALLTALK, C++, POOL
Functional Programming	
data flow	ID, LUCID, VAL, VALID
applicative	Pure LISP, ML, MIRANDA
Predicate Logic Programming	PROLOG, GHC PROLOG
Production System Programming	OPS5, LOOPS
Semantic Network Programming	NETL, IXL
Neural Network Programming	P3, NIL
Application-Oriented Programming	
Spreadsheet	VISICALC, LOTUS 1-2-3
Relational Database	Dbase3
Robotics	AML, VAL

Treleaven said that conventional languages are the only class of programing languages of which most users of computers are familiar. This class has developed for programing the traditional von Neumann stored program computer. Hence, the semantics of conventional languages reflect the von Neumann programing model: global memory, fixed-size memory cells, assignment, and sequential execution. Concurrent (procedural) languages extend this control flow programing model with parallel control structures based on processes, plus communication and synchronization mechanisms. A process is an independent program consisting of a private data structure and sequential code that can operate on the data. Concurrently executing processes operate on their own private data and only interact with one another using the communication and synchronization mechanisms. The communication mechanism is the way processes communicate data among themselves. The most commonly employed mechanisms are: unprotected shared (global) memory, shared memory protected by modules or monitors, message passing, and the rendevous. The synchronization mechanism is the way processes enforce sequencing restrictions among themselves. The commonly employed mechanisms include: signals, synchronized sending, buffers, path expressions, events, conditions, queues, and guarded regions. In general, concurrent procedural languages may be classified by the nature of their communication mechanism into: shared memory (for example, ADA) and message passing (for example, OCCAM).

Shared Memory. Treleaven said that in these concurrent procedural languages, communication of data is by shared memory with concurrent access being controlled by the synchronization mechanism (for example, signals, events, monitors, rendevous). Examples of concurrent (shared memory) languages include: ADA, MODULA, MODULA-2, Concurrent PASCAL, Path PASCAL and PL/1.

ADA, the US Department of Defense language, is designed for programing embedded computer systems – large real-time systems which inherently involve concurrency. An ADA program is written as a number of quasi-independent sequences of statements called Tasks (i.e., processes). ADA uses a main PROCEDURE where, in contrast, PASCAL uses a PROGRAM. A PROCEDURE consists of one or more Tasks. Each Task must have a specification part and a "body" containing the local data and code.

Thus, in ADA, parallel programing is based on Tasks and also the rendevous, implemented by "entry" and "accept" statements. The essential idea of the rendevous between two tasks is the synchronized transfer of information at a predefined point in each process. The corresponding "entry" and "accept" statements in a task are specified as for a procedure declaration, including parameter lists. An "entry" specifies the name and parameters of a service which can be called from elsewhere in the parent unit (normally in a sibling task). An "accept" statement has the syntax of a procedure (without local data declarations) appearing embedded within executable statements, but executes as a critical region.

The entry/accept service works by the calling task executing an entry call statement – similar to a procedure call – signalling the wish for a rendevous. The task containing the entry/accept signals its willingness to provide the service by reaching the "accept". When both calling and accepting tasks have arrived, the rendevous occurs. During the rendevous, the parameters are transferred to the accepting task and the calling task suspends execution. The body "do-end" of the "accept" is then executed as a critical region. Once the "accept" statement terminates, its parameters are returned to the calling task, and both tasks resume independent execution until another rendevous is attempted. Treleaven said that although it can be claimed that this form of shared memory parallelism is the most natural extension to conventional languages, it seems to have little to offer to the design of languages for neural networks.

Message Passing. In concurrent (message passing) languages such a OCCAM, messages are used to handle synchronized communication between parallel processes, with shared memory being used for state within a process. With message passing, data is passed directly, using a channel or queue from the transmitting process to the receiving process, which stores the data locally in its private store. In OCCAM, as well as the traditional control structures (for example, IF and WHILE) there are also control structures for sequential (SEQ), parallel (PAR), and alternative (ALT) process execution. Examples of these message passing types of programing languages include CSP (Treleaven and Gouveia, 1982) and OCCAM, as well as GYPSY, PARLANCE, and PLITS.

OCCAM, originating from Hoare's CSP (communicating sequential processes), is based on processes (PROC) which may execute concurrently and communi-

cate using channels (CHAN) as shown in Figure 3. The most direct implementation of an OCCAM program is a network of microcomputers each executing a process concurrently. However, the same program could also be implemented by a single time-shared processor, according to Treleaven.

```
PROC p1 (CHAN c11,...,c1n) =          "declaration"
  VAR v1,...vm :
      . . .

PROC p2 (CHAN c21,...,c2n) =          "declaration"
  VAR v1,...vm :
      . . .

PROC network (CHAN in, out) =
  CHAN c1,..,cn:
  PAR
    p1(c1,...,cn)                  "interconnection"
    p2(c1,...,cn)
```

Figure 3. Program structuring in OCCAM.

A process – the fundamental working element in OCCAM – is a single statement, group of statements, or group of processes. Programs are constructed from three primitive processes: assignment, output, and input. Assignment "x: = y" sets the value of a variable to an expression. Output "c!y" is used to output a value of an expression "y" to a channel "c". Input "c?x" sets the value of a variable "x" to a value input from a channel "c". A channel is an unbuffered structure that allows information to pass in one direction only, synchronizing the transfer of information. Thus, a channel behaves as a read-only element to a receiving process and a write-only element to the transmitting processing. The transmitter can only write when the channel is empty, while the receiver can only read when the channel is full.

To control the order of execution of such processes OCCAM provides three control structures: sequential (SEQ), parallel (PAR), and alternate (ALT), as well as the traditional IF and WHILE constructs. "ALL" controls precede a list of processes, SEQ and PAR defining sequential and parallel execution, respectively. ALT causes exactly one of a list of processes to be executed, and will wait until at least one of the "guarding" conditions is true.

To summarize, Treleaven said that in OCCAM a parallel program is represented as a network of communicating processes and therefore already contains many of the mechanisms required of a neural network language. In addition, OCCAM contains useful constructs for specifying the replication of networks of processes and channels. On the negative side, OCCAM does not currently allow processes and channels to be created dynamically. However, this is not precluded by the semantics of the language. Another point, Treleaven said, is that OCCAM does not directly support the concept of a branching axon, where a single output broadcasts to a set of input channels.

Object-Oriented Programing. This type of programing is an attempt to generalize the concurrent language concepts of processes and message passing. Object-oriented programing centers on the concepts of object, class, and instance. An object (cf. module or process) is an active system component consisting of some private memory and a set of operations. Communication between objects is via messages, a message being a request for an object to carry out one of its operations. A class describes the implementation of a set of objects that represent the same kind of system component. The individual objects described by a class are called its instances. An object's private properties are a set of instance variables that make up its private memory and a set of methods that describe how to carry out its operations.

Ingalls (1978) uses the following example to distinguish between Object-Oriented programing and PROCEDURAL programing:

> "some object + 4" means to present "+4" as a message to the object. The fundamental difference is that the object is in control, not the "+". If "<some object>" is the integer "3", then the result will be the integer "7". However, if <some object>" was the string "ABCD" the result might be "ABCD4".

To fully support Object-Oriented programing, a language should embody four properties, according to Treleaven:

- Information hiding – where the state of the object is contained in its private variables, visible only from within the scope of the object
- Data abstraction – allows a programer to define abstract data types, each consisting of an internal representation plus a set of procedures used to manipulate the data
- Dynamic binding – where the type (for instance of a message) can only be determined at run-time and not from the program text
- Inheritance – allows classes, and hence objects, to be created that are specializations of other objects, inheriting the variables and methods of the superclass.

Examples of Object-Oriented languages include SIMULA (the first language to explore Object-Oriented programing), SMALLTalk, C++, and POOL (one of the few parallel languages).

According to Treleaven, Object-Oriented programing may be viewed as a more sophisticated form of program structuring to the communicating processing model in languages such as OCCAM. Whereas in the late 1970's and early 1980's, processes were the major concept used for parallel programing, now, in the late 1980's, objects are taking over that role. Thus in any very-high-level neural network language, objects would seem to be the natural concept for program structuring, with mechanisms for information hiding, data abstraction, dynamic binding, and inheritance.

Functional Programing. This program operates by regarding a "program" as a collection of functions in the true mathematical sense. A function is applied to its

(input) agreements, which results in output values, and may for example, define complex operations over list-structured arguments. Functional programs possess some important properties not usually found in Procedural styles. For instance, "referential transparency" ensures that the value of an expression is determined only by its definition and not by its computational history. Freedom from side-effects and the restriction of assignment operations also contribute to a more "pure" model of computation, according to Treleaven. He then discussed two important classes of functional programing languages: Data Flow language and APPLICATIVE languages

Data Flow Languages. In the Data Flow programing model, a statement outputting (i.e., producing) a result passes a separate copy to each statement wishing to input (i.e., consume) the value. There is no concept of variables in a Data Flow program; data is passed directly from one statement to another. Execution of a statement is "data driven," with a statement being executed as soon as all its input values are available. Statements in Data Flow languages follow a single assignment rule: a name may appear on the left side of an assignment only once within the area of the program in which it is active. For the control mechanism, although execution is data driven, statements in Data Flow language are superficially similar to statements in conventional languages. Examples of Data Flow languages include ID (Arvind and Gostelow, 1982), VAL (Ackerman, 1982) and VALID.

In Data Flow languages an important property is the copying semantics, which means that any operation on a data structure always creates a new structure. For instance, an array is not modified by a subscripted assigment statement "array[(index)]: = value" but is processed by an operator which creates a new array. In the ID language this operator appears as:

new array←array + [index] value

while in the VAL language it is written:

new array : = array [index:value].

These single assignment statements are adequate for simple assignment of the form: name: = expression, and even for conditional statements, as long as they are constrained to conditional expressions: name : = if expression then expression else expression.

According to Treleaven, the advantages of Data Flow languages are that their single assignment syntax is similar to conventional languages, that single assignment languages are used to specify directed graphs, and that parallelism is implicitly expressed. With regard to neural networks, the data-driven model is close in concept to the firing of a neuron when new inputs are available. However, the data flow model specifies that a data value must be available on each of the inputs, in contrast to a neural model, which requires only a subset of the inputs to be available before a neuron can fire.

Applicative Languages. Applicative languages are so called because of the dominant role played by the applications of functions to structures. Treleaven said that the important notion associated with Applicative languages is that the value of an expression (its meaning) is determined solely by the value of its constituent parts. Thus, should the same expression occur twice in the same context, it denotes the same value at both occurrences. A language having this property for all its expressions is referred to as an Applicative language. There are a number of interesting Applicative languages, according to Treleaven, such as Pure LISP (Winston and Horn, 1981), SASL (Turner, 1979), and MIRANDA.

In the SASL system, for example, a program is a collection of equations by means of which the user attaches names to various objects. There are four types of objects: numbers, strings enclosed in double quotes, lists, and functions. Numbers and strings have the normal properties expected, with the ususal kinds of operations defined on them. Lists are written using round brackets and commas: number = (1,2,3,4,5,6,7,8.9,10) and elements of a list are accessed by indexing. For example, the expression "number 3" would here give the result "3".

Functions are denoted by writing down one or more equations with the name of the function (followed by some formal parameters) on the left and a value for the function on the right. For instance, the obligatory factorial is expressed as shown in Figure 4. The order in which equations are written has no logical significance. Where order is important a boolean "guard," such as "n>0" (Figure 4), is placed in front of an expression. More sophisticated forms of pattern-matching involve the use of list structures in formal parameter positions.

```
DEF

     fac 0 = 1

     fac n = n > 0 -> n * fac(n - 1)

?
```

Figure 4. Program structuring in SASL.

Treleaven said that Applicative languages have a number of good programing properties such as: (1) the uniformity of the structures manipulated, (2) implicitly expressed parallelism, and (3) the absence of side-effects and explicit sequential execution. However, it is not clear how powerful these mechanisms will be for programing neural networks.

Predicate Logic Programing. Logic Programing in languages such as PROLOG is based on the Horn Clause subset of predicate calculus, where the basic concepts are: statements are relations of a restricted form, and execution is a suitably controlled logical deduction from the statements.

For many applications of logic it is sufficient to restrict the form of clauses to those containing at most one

conclusion. Clauses containing at most one conclusion are called Horn Clauses, after the logician Alfred Horn. Each clause is either an assertion or an implication. In general, every assertion is an atom "A", whereas every implication has the form "A if B1 and B2...and Bn" and all conclusions "A" and conditions "B1, B2,...Bn" are atoms, expressing a simple relationship among individuals.

PROLOG is the predominant Logic programing language, with a number of parallel variants appearing such as PARLOG (Clark and Gregory, 1981), Concurrent PROLOG (Shapiro, 1983) and GHC PROLOG (Ueda, 1985).

According to Treleaven, the main features of Logic programing are pattern-matching (unification) and substitution. He said that the advantages of Logic programing include the fact that it is the most "high level" programing model, in specifying "what" rather than "how" a computation is to be executed, and the programing style is close to knowledge-based systems. Disadvantages of Logic programing are that the notation is very concise and therefore can be difficult to understand when seen in the form of a program. As with the related Applicative languages, Treleaven said that it remains unclear if Logic languages are related to neural networks.

Production System Programing. These languages are primarily concerned with the representation of human knowledge and mechanisms of inference from such knowledge. With Knowledge-Based languages, also referred to as Production System or Rule-Based languages, the knowledge base consists of rules in the form of "condition action" pairs, as follows:

```
RULE r1 IF condition1 AND condition2
AND...THEN action1  AND action2  AND...RULE r2.
```

These rules may be used in two ways, either reasoning forwards from a condition to an action, or reasoning backwards by hypothesizing an action and using the rules to verify the condition. Treleaven said that Knowledge-Based languages have been used extensively for a variety of tasks including research in artificial intelligence, cognitive psychology, and learning systems.

A Production System typically consists of three parts, namely, the Working Memory, the Production Memory and the Inference Engine. The Working Memory is a global data base of symbols representing facts and assertions. The Production Memory stores the set of rules constituting the program. Lastly, the Inference Engine controls the execution of the rules.

The Inference Engine, during execution, attempts to match the patterns in the Productions with data elements in the Working Memory, comparing the subelements of each structure. Any variable bound to the element it matches is a rule.

Treleaven said that OPS5 and other Production System languages are functionally complete languages; their in-built capabilities are sufficient to compute any compatible function (subject to resource limitations) without departing from the production system formalism. However, Production System languages seem to share with Applicative and Logic languages, a "pattern-matching" form of execution which is difficult to equate with the neural network type of computation.

Application-Oriented Programing. Treleaven said that he was using this type of programing in his talk to cover languages developed for specific application areas. In these application areas, the boundary between languages and certain software packages or utilities is cloudy. Example areas include financial-modeling, relational data base, and robotics.

As an example of application-oriented lanaguages, Treleaven examined the spreadsheet languages, designed for financial modeling, but also used in the additional fields of engineering, science, education, statistics, etc. He said that in any field where tabular reports of rows and columns of calculated numbers are required, the spreadsheet language provides a very powerful tool. Examples include VISICALC, MULTIPLAN, and LOTUS 1-2-3.

According to Treleaven, these application-oriented languages appear not to be particularly useful in the design of a neural network language insofar as we consider neural networks a fundamental new domain of computation rather than just another new application area. He said that, in contrast, semantic networks are very close to neural networks.

Semantic Network Languages. Treleaven said that semantic networks represent one end of the Connectionist spectrum with neural networks at the other end. Thus, semantic networks and neural networks are closely related, and any flexible neural network language may also be able to represent semantic networks.

Overview. A semantic network is basically a directed graph in which the nodes represent independent entities which may be concepts or objects, and the directional links represent the relationships between these entities. In such a network, information is stored not only in the nodes but (just as importantly) also in terms of the interconnections among the nodes. Adding new information into the network usually involves creating new nodes or changing the interconnections among the nodes. These networks can perform searches and simple inferences very effectively and with great parallelism, according to Treleaven. One of the fundamental operations of these networks is the propagation of simple (for example, 1 bit) markers along the interconnections. It is the sumultaneous propogation of markers along many interconnections which gives this type of architecture the major source of parallelism. This type of organization is sometimes known as "marker-passing" parallel architecture. Since connections play such an important role both in information storage/retrieval and processing, semantic network models fall into the spectrum of connectionist models.

Treleaven said that there are a number of well-publicized semantic network languages. One example is the

NETL system, which is a simulation of a direct hardware implementation of a semantic network. Knowledge is stored in the network in terms of symbolic assertions. The system was designed to store/access a large number of assertions quickly, and to perform inferences and searches in a highly parallel manner. Another example is the Japanese IXL language.

IXL. The semantic network language IXL was designed at the Electrotechnical Laboratory in Japan for programing the IXL semantic network machine. IXL can perform description of knowledge bases, modification, and search operations using semantic networks. In IXL, semantic networks are used to represent declarative knowledge, and logic-like expressions are used to represent procedural knowledge.

The main features of IXL are shown in Figure 5. These are: (1) Relations in IXL are classified into basic hierarchical relations (is a, instance of) and general relations [general relations are further classified into assertion and properties]; (2) a general relation is represented not by a link but by a node in order to precisely define the meaning of the relation; (3) procedural knowledge is described in logic-like expressions; and (4) knowledge can be checked for inconsistency by the a kind of link.

```
To construct a relation:
   assertion (R, X, Y).
   property (R, X, Y).

To inquire about a link:
   isa (X, Y).
   instance (X, Y).
   ako (X, Y).
   source (R, X).
   destination (R, Y).

To inquire about a relation:
   asst (R, X, Y).
   prop (R, X, Y).

To connect nodes by a link:
   link (is_a X, y).
   ...
   link (instance_of X, Y).
   ...
```

Figure 5. IXL Semantic network language.

Relations in IXL are classified into basic hierarchical relations supported by the system and general relations which are defined by the user to specify his application program.

General relations are divided into assertions and properties. When the user defines a general relation, it is necessary to declare to what the relation belongs. For example, when a concept X has a relation R within Y, X is called the source of this relation and Y is the destination.

Treleaven said that IXL is a textual semantic network language, but the user also has access to a graphical system that can be used to specify and debug a knowledge base. Networks specified in the graphical notation are translated into IXL for interpretation. IXL is implemented in DEC-10 PROLOG. Further details on the IXL language can be found in Higuchi's paper (Higuchi, 1985), which also contains details of the IX machine designed to execute IXL.

Treleaven said that programing languages for neural networks still remains one of the least developed areas of neurocomputing. There are a few systems available, ranging from environments to languages, that can be used for programing neural networks. These include: P3, which runs on a Symbolics 3600; ANNE, a simulator running on the Hypercube; GINNI, from SAIC of Tucson, Arizona; AXON, from Hecht-Nielson Neurocomputer Corp., California (Hecht-Nielson, 1987); and CONNARTIST which is a slightly modified version of P3 with a few more graphics possibilities and NIL programing language developed at the University of London, UK.

P3. The P3 language was developed by Zipser and Rabin (1986) as a simulation tool to aid in the development of the Parallel Distributed Processing (PDP) group's models. The motivation was that it is easy to code the basic PDP algorithms, but that the user interface and debugging tools are difficult and time consuming to develop. P3 is written in LISP and runs on a Symbolics 3600. Although it was not originally intended as a neural computer language as such, it has all of the essential pieces and has an inherent parallel structure that would map onto suitable parallel hardware, according to Treleaven.

The principle components of the P3 system are the *plan language*, the *method language*, the *constructor*, and the *simulation environment*. The *plan language* is a connection description between each of the nodes (called units in P3), and a definition of the types of units which will make up the model. Each type of unit has an associated method or modified LISP program associated with it. This modified LISP language, which implements the internal computational behaviors of the units in a model is called the *method language*. After the plan and the associated methods have been specified they are built into an appropriate data structure by the *constructor*, which acts in many ways like a compiler. This data structure is read into the *simulation environment* where two layers of debugging tools can be used to test the code, and the simulation can be run. The simulation provides the user with a display of the nodes in the user-chosen orientation. It uses the "window" and "mouse" environment of the Symbolics 3600. The debugging tools allow each of the expected connections to be interactively tested, and allow "strip chart recorders" to be connected to any of the par-

ameters of a unit so that the time behavior of that parameter can be measured.

The environment of the network is handled by an appropriately defined environment that handles any input or output connections, and has a suitable defined method. Control over the update sequence of units is also handled in this manner via a control unit. Without this control unit each of the units will be sequentially executed, but through this unit an asynchronous updating process can be simulated. Treleaven thinks that programs like P3 will be useful in the early stages of model development when the size of the models are modest and there is frequent need for changes in structure.

ANNE. ANNE (Another Neural Network Emulator) is a neural network simulation system, developed at the Oregon Graduate Center for the Intel iPSC. This system consists of a compiler that takes as input a Network Description Language (NDL) and generates: (1) a low-level generic network specification called the Beaverton Intermediate Form (BIF); (2) a network partition utility for mapping the neural net onto the iPSC; and (3) a dedicated ANNE emulator for the iPSC. A user describes his neural network in NDL, then specifies the nodes and combines them into a network. The functions of the nodes for learning and computation are specified by pointers to C procedures. Internal to these C functions, user code accesses network parameters via standardized data structures and calls to ANNE.

The NDL source is compiled into BIF and run through the mapper which, using an Intel iPSCPAD (Physical Architecture Descriptor) file that described the target machine, partitions the network among the iPSC processors. This mapped BIF is then read by ANNE, and used to construct a network emulator.

The purpose of ANNE, Treleaven said, is to act as a testbed and debugger for the variety of neural network models describable by BIF. ANNE is not intended to model any particular architecture design on which these networks might be mapped, but rather, it is designed to run neural networks in an expedient manner in order to examine their operating characteristics. As part of the complete design environment ANNE supplies: (1) a scanner/builder to construct a neural network from BIF; (2) a message-passing mechanism for coordinating communication between network nodes on different iPSV processors; and (3) a special timing and synchronization scheme which is user controlled. In addition, the user has the ability to examine, modify, or save pertinent data within the network, including the entire BIF specification (plus network state) of the network at any point in the simulation.

GINNI. The GINNI system, developed by Scientific Applications International Corporation (SAIC), is an interactive development environment for neural networks. GINNI provides for neural network creation, geometry configuration, and equation editing, and a compiler and command interpreter for execution hosted by a Symbolics 3600.

A high-level network description language specifies equations and connections, on a layer-by-layer basis. For each layer the user may define arbitrary activation equations, learning rules, nonlinearities, and constants. Once a network has been defined and connected, the user may execute all or part of the specified model in either interactive or batch mode.

The GINNI system has a capacity for 5 million processing elements (i.e., artifical neurons) and 10 million connections. Weighted connections are processed at a rate of 35,000 per second. These numbers are for the Symbolics 3600 and 160 Mbytes of swap space. GINNI is supplied with three neural network models which can be used directly or as templates for custom networks. These models are (1) Adaptive Resonance, (2) Backpropagation, and (3) Hopfield.

UCLA SFINX. Structure and Function in Neural Connections (SFINX) is a neural network simulator environment, developed at the Machine Perception Laboratory of the University of California, Los Angeles (UCLA) that allows researchers to investigate the behavior of various neural networks.

As shown in Figure 6, the SFINX structure is analogous to traditional languages. A neural network algorithm is specified in a high-level textual language. This is compiled into an equivalent low-level language. Next, the low-level language is assembled into a binary data structure. Lastly, this data structure (defining the network) is loaded into the SFINX simulator for interactive execution.

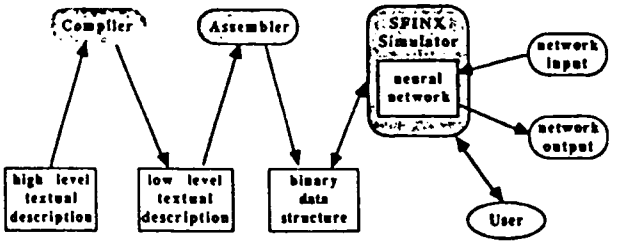


Figure 6. UCLA SFINX environment.

In SFINX, network specifications have two basic parts: (1) set of nodes – a node is a simple computing element composed of memory [storing the state of the node] and functions [defining how signals are processed]; and (2) interconnections – defining the connectivity and the flow of data among the nodes.

These network specifications are represented by virtual PE's, each comprising: (1) functional pointer, (2) output register, (3) vector of state registers, and (4) vector of associated weight/link address(es). Lastly, the front-end of the SFINX simulator is a command interpreter, accepting SFINX shell scripts. These shell commands include: *load, peek, poke, run, draw*, and *ser*. Once a network structure is created, these SFINX shell com-

mands can be used to exercise the simulator, displaying and modifying the state of the network.

IBM CONE. The IBM Computational Network Environment (CONE) (Hanson, 1978) programing environment comprises: (1) the high-level General Network Specification Language (GNL), (2) a general intermediate network specification (NETSPEC), and (3) an Interactive Execution Program (IXP). As illustrated by Figure 7, neural network programs are specified in GNL and compiled into the machine-independent NETSPEC. Then an assembler translates a NETSPEC into a "Network IXL" for execution on a specific neurocomputer simulator or emulator.

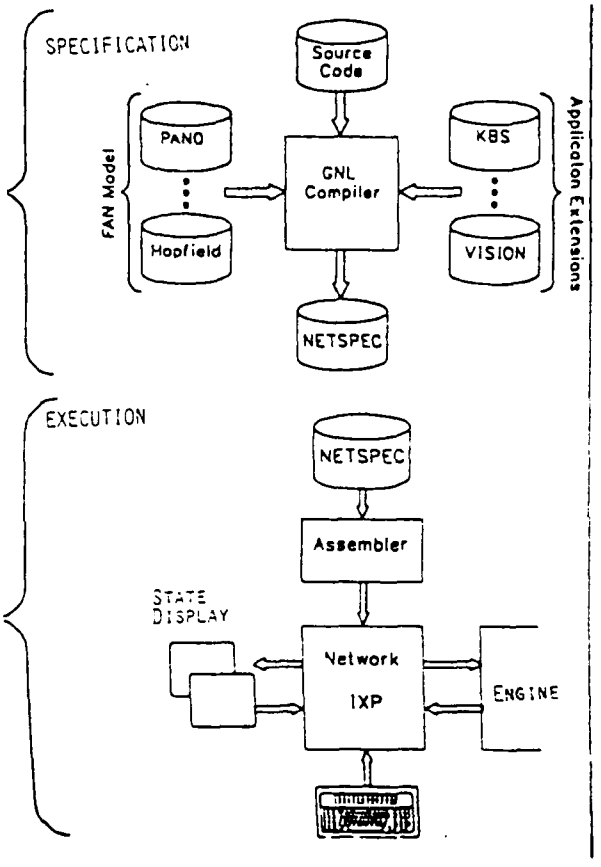


Figure 7. IBM CONE environment.

In the GNL, network specification centers on hierarchy and function decomposition. The main elements of GNL are: (1) proc – the functional processing elements, (2) ports – the inputs and output ports of a processor, and (3) paths – the communication links between processor's ports. The GNL compiler takes the source statements together with application-specific extensions to the compiler, and produces a NETSPEC generic intermediate network specification. This description is simply a parts list of the various PE's and a list of their topology.

The NET Interactive Execution Program (NET IXP) runs on a PC connected to the Network Emulation Processor (NEP). The IXL has three main components: (1) the graphics display, (2) the operator interface, and (3) the network engine. The graphics display shows the state of the network, with the operator selecting the set of procs of interest and also the level of hierarchy. The IXL operator interface is a command shell which controls the execution of the network engine. Lastly, the IXL network engine provides a simple interface to the target neurocomputer.

NIL. The goal of the Neural Network Implementation Language (NIL), developed at University College, London, UK, is to program a range of neural network algorithms and map them onto a range of neurocomputer architectures. NIL is intended to cover the spectrum of connectionist models, from semantic networks to neural networks. It therefore supports the dynamic creation and deletion of nodes and interconnections required by semantic networks. The language comprises two parts: (1) basic part – for defining the functions of the nodes and the organization of the network, and (2) manipulation part – for modifying the network, for monitoring its state, and for interacting with the environment. Central to the design of the basic part is the "guarded process" construct: (input condition = > (statement) where the input conditions define the set of inputs that must be available before the corresponding statements is executed. An input condition consists of a subset of inputs, all of which must be available for the corresponding statements to be executed. Statements comprise Dijkstra's guarded commands (Dijkstra, 1975), namely, IF, DO, ASSIGNMENT, and SKIP.

A network is specified, first, by giving function definitions for the different types of node, and secondly by giving link definitions of the interconnection network. A "fun" definition gives the function name, the vector of inputs, the vector of weights associated with the inputs, and the vector of the outputs. Once the node functions are defined, the neural network interconnections are given by link statements. A link statement specifies the function name, a list of inputs, a list of initial values for the inputs, and the list of outputs. Replication statements are also provided for specifying the replication of similar link statements.

The "manipulation" part, the second part of the language, performs the following tasks: (1) reads the values on the links of the network, (2) feeds new information into the network, (3) establishes and deletes links, and (4) reads the state of each node in the network. The manipulation part of a program has similar constructs to the basic part, being based on guarded commands.

Future Trends

Treleaven said that traditionally, each new class of parallel computers has led to an associated class of high-level languages embodying the same model of computation. An example is the Transputer and OCCAM, both based on communicating sequential processes. Another example is Reduction machines and Functional languages.

Although programing systems for neural networks are still at an early stage of development, a number of trends are discernible, according to Treleaven. Most of the current generation of neural network systems are software simulators that have highly sequential features. From such simulators, complete programing environments are being developed such as ANNE, SFINX, and CONE. The neural network programing environments typically comprise (1) a high-level textual language, (2) a low-level machine-independent network specification language, (3) a machine-dependent interactive network simulation system, and (4) associated software tools such as graphical displays of networks.

According to Treleaven, it is to be expected that this trend in programing environments for neural networks will continue. The high-level languages will be enhanced for parallelism by absorption of programing concepts from other classes of languages such as object-oriented languages. In addition, Treleaven expects that a standard low-level generic network specification language (for example, NIL or BIF) may emerge. However, it is unclear where it is possible to develop any standard in the area of imteractive network simulation systems, given the very wide range of neurocomputer architectures currently under development.

Associative Memories and Representations of Knowledge as Internal States in Distributed Systems

This topic was discussed by T. Kohonen (Laboratory of Computer and Information Science, Helsinki University of Technology, Finland). Two of the main aspects of biological memory are the structure of internal representations of knowledge in the neural network and the memory mechanism itself. The questions concerning the former must be settled before the latter can be approached. Therefore, in his talk, Kohonen discussed "self-organizing maps" of the input signal space which constitute the information to be stored in a generalized distributed associative memory, and then spoke about distributed associative memory.

Preliminary Remarks. Kohonen said that the distributed, correlation-matrix-feedback type associative memories have recently been greeted with great enthusiasm. However, he said that the following aspects need to be emphasized:

1. Associative memories have been implemented by digital techniques since 1955. Compared with the exemplary cases recently discussed in "neural network" literature, the digital content-addressable memories outperform the neural ones by orders of magnitude in capacity, speed, stability, accuracy, and selectivity of recall. The only merit of the latter to be studied further is spatial distributedness of the memory traces, which makes the "neural" memories resistant to local damage, like a hologram.

2. Many "connectionist"-type distributed associative networks are not genuine memories because their interconnection strengths must be precomputed from the signals and loaded onto the network. For instance, in the Hopfield model, the feedback matrix is computed from the wanted outputs.

3. There also seems to be a certain amount of confusion concerning the original ideas. Kohonen stated that Anderson was the first to devise the principle applied by Hopfield. However, a genuine associative memory, the internal state of which is directly determined by the received input siganl patterns, was published earlier by Kohonen. These networks have the same structure (matrix feedback) and function (correlation matrix) but there are no conceptual difficulties of the kind shared by Anderson and Hopfield.

According to Kohonen, one has to realize that the two main problems connected with biological memory and "intelligent" memory of artificial neural networks are: (1) what kind of internal representations of input signals (and representations of knowledge) are formed onto the neural network before anything is stored and (2) what is the distributed associative memory mechanism itself. According to Kohonen, most of neural network research only seems to concentrate on the latter question.

A Typical Structure in Neural Networks. The biological brain consists of many anatomically distinguishable formations, each one with a definite physiological function. There exist certain characteristic differences between the network structure and functions of the various formations (Shepherd, 1974). Nonetheless, the structure depicted in Figure 8 seems to be rather typical for many parts – the neocortex, for example, and the hippocampus. The large symbols stand for neurons (actually, principle neural cells) and the lines represent neural fibers, the axons; the small circles stand for the variable junctions called synapses. Kohonen said that it is already possible to demonstrate several interesting information processing phenomena in the basic model of Figure 8. In particular, Kohonen spoke about two of them, namely, self-organized formation of ordered internal representations and associative memory. He said that more natural behavior is expected to ensue from a higher sys-

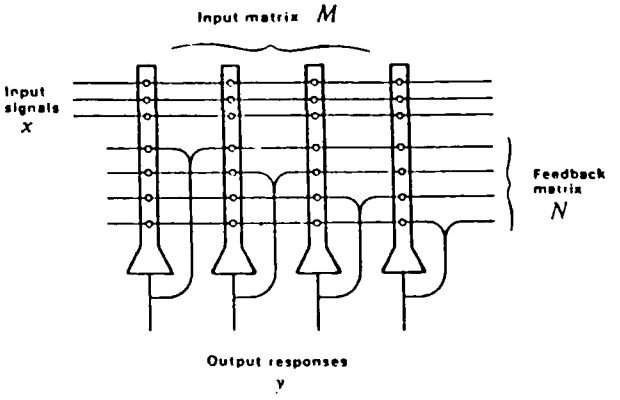


Figure 8. The basic structure of neural networks.

tem organization in which several networks of this type are interconnected.

Self-Organizing Maps of the Sensory Signal Space

Kohonen said that certain attempts to simulate the organization of the visual cortex had already been made in 1973; it turned out later, however, that it is not easy to achieve a high self-organizing power in neural networks unless the control of neural activity is made in a more specific way. This finding, first reported by Kohonen in 1981 (Kohonen, 1981; Kohonen, 1982) has been described in detail in the latest book by Kohonen (1988). For brevity, only a shortcut algorithm, derivable from the biophysical process (Kohonen, 1982) and readout amenable to computation by contemporary hardware was presented by Kohonen (Figure 9). In this figure where a two-dimensional neural network is shown schematically, every neuron or node has its own parameter vector, m_i, eventually identifiable with the input weight vector of a formal neuron. Kohonen said that it is assumed that by virtue of the underlying detailed neural network structure, which need not be known here, the following two partial operations are implementable:

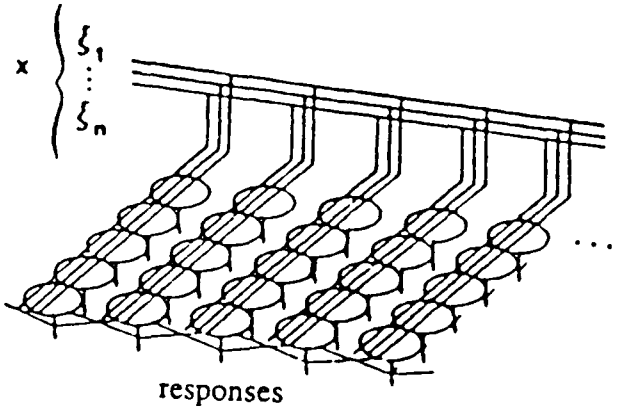


Figure 9. Network for self-organizing maps.

1. It is possible to compare an n-dimensional signal vector, x, with all the weight vectors, m_i, in parallel and to identify that node c, the weight vector m_c of which matches best with x – with respect to some similarity measure (metric). This node then defines around it an *active neighborhood*, N_c, which contains all the nodes within a certain radius from node c.

2. The weight vectors within N_c, irrespective of their value, can adaptively be changed towards x such that their *similarity* with x is increased.

Kohonen said that the self-organizing process defined by these rules seems to be to a great extent independent of the choice of metric: computational reasons may favor a particular one. If Euclidean metric is used, the above rules can be dressed into the following algorithm form. Assume that $x = x(t)$ and $m_i = m_i(t)$ are functions of integer-valued time, t. Further it is assumed that the initial values, $m_i(0)$, are random vectors.

Matching: $||x(t)|| = \min(||x(t)-m_i(t)||)$. Index c defines the center of the active neighborhood N_c, and a suitable radius for it can thereafter be set, as described below.

Updating: $m_i(t+1) = m_i(t) + \alpha(t)\,[x(t)]$ for $i \in N_c$, $m_i(t+1)$ otherwise. Here $0 < \alpha(t) < 1$ is a scalar valued "adaptation gain" which is a Monotonically decreasing function of time.

Kohonen said that in order that the network would operate as a signal processor, each node should also produce an output response. Such responses do not enter the self-organizing algorithm, and so it is possible to define the response as any function of x and m_i. For instance, the inner product, $m_i^T x$, or some "sigmoidal" function of this value can be used in a technical implementation. Kohonen said that we may also assume that an active response is only obtained from the best-matching node c used.

Kohonen said that the choice of $N_c = N_c(t)$ and $\alpha = \alpha(t)$ is not unique; many different functional forms will guarantee the following results. Kohonen and his group have tried forms for $\alpha(t)$ which are inversely proportional to t, and exponentially or linearly decreasing with t. The last choice is simplest. Nonetheless, Kohonen said that it is advisable to distinguish between the initial ordering phase, during which the m_i attain their correct topological order, and the final convergence phase, during which the exact asymptotic values of m_i are "fine-tuned." The former phase may take, for example, 1000 iteration steps, whereby $\alpha(t)$ may decrease linearly from a rather large value, such as 0.9 to 0.01, and the radius of $N_c(t)$ may decrease linearly from, say, half of the diameter of the network to one spacing. During the final phase, $\alpha(t)$ may decrease linearly from 0.01 to zero, while the radius of N_c is one spacing. The length of the final phase depends on the desired accuracy, but in general, it is significantly longer – a multiple of the initial one.

Kohonen said that it is rather difficult to describe analytically what happens in the process defined by the equation shown above, for matching and updating. Qualitatively, one can state , for instance, that because m_i are always changed as chunks consisting of several neighboring units, their value as a function of the network coordinates tends to become smoothed, and very apparently also ordered sooner or later. However, Kohonen said that the boundary effects are very subtle and difficult to understand. On the other hand, it also seems obvious, according to Kohonen, that the m_i, or their distribution, tend to imitate x, or its statistical density function. In other words, some kind of ordered image of the density function of x will be formed into the network, into the set of values m_i. The images of different x values will further become ordered in such a way that the topology of the corresponding nodes in the network tends to be the same as the topology of the corresponding signal values of x.

Simulations. Kohonen said that the self-organizing result, which seems to emerge like *deus ex machina* from the process can be illustrated with practical examples. Kohonen said that he and his group have performed

demonstrations on a dozen or so maps of very different kinds; the most advanced ones are those which have been used to recognize phonemes from continuous natural speech. The following simple example, presented by Kohonen, demonstrates a rather biological-looking function which combines several different sensory "channels" in the same map. In other words, the internal representation of the environment in the map is redundant with respect to the different sensory modalities.

The network is of the type shown in Figure 9. The input vector to it shall be six-dimensional, and its components shall be derived from the system model depicted in Figure 10 in the following way. Two of the signals correspond to two coordinates ($\xi 5$ and $\xi 6$) which indicate the rotation of an artificial eye. The gaze of the latter shall be directed to a moving point on the plane. Two pairs of coordinates, ($\xi 1$, $\xi 2$) and ($\xi 3$, $\xi 4$), indicate the bending angles of two artificial arms. During learning, these arms shall touch the same point at which the gaze is fixed.

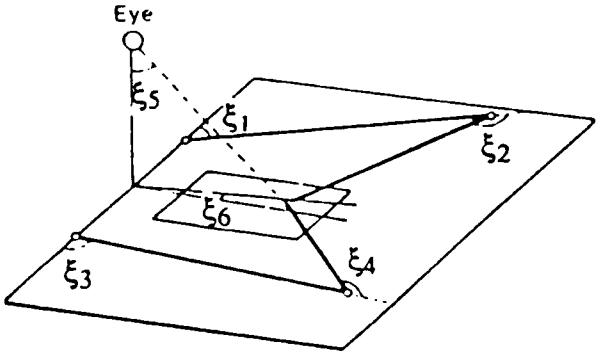


Figure 10. System model for mapping the environment through several parallel sensory channels.

When the target point is moving randomly over the framed area on the plane, a stochastic signal vector, x = ($\xi 1$, $\xi 2$, $\xi 3$, $\xi 4$, $\xi 5$, $\xi 6$), is obtained, the subsequent values of which are applied to the self-organizing algorithm. The various nodes will then be sensitized to different targets on the same plane such that each of the latter selects a different unit c of the neural network in an orderly fashion. Figure 11 shows a network of lines drawn onto the said plane. Each crossing and corner point corresponds to a "due" node in the neural network which is then selected with these particular coordinates – i.e., which "responds" to this point. The test was made for each hand and the eye separately, by disconnecting the coordinate signals from the other two organs; the three maps (Figure 11) are practically identical, showing that the representation is triply redundant, according to Kohonen.

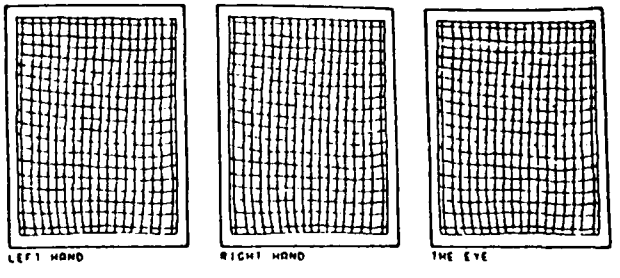


Figure 11. Target plane of figure 3 and its calibration for the tree channels.

Distributed Associative Memory

Kohonen said that it seems that most "connectionist" models implicitly assume some kind of localized representations of stored items associated with the nodes of the network. Frequently also, a distinct semantic value is assigned to each node. In a genuine, autonomously operating neural network, however, such a specificity cannot be preassumed, and in a massive network, it would even be technically impossible to load all the nodes with such a meaning, according to Kohonen. He said that assumption of an underlying control system for the routing of signals to the nodes would be absurd and would lead to a vicious circle in theoretical discussion.

The above self-organizing mapping principle discussed by Kohonen, on the other hand, is directly able to form localized, spatially encoded representations of the input signals, as illustrated by the previous example and numerous other simulation experiments performed by Kohonen and his group. He said that one has, therefore, to regard such activity distributions over the network as the patterns to be stored and recalled. "Knowledge," he said, might then be represented by a plurality of such maps, each one describing the encoding of particular sensory signals and possibly their higher aspects.

The mechanism of autoassociative recall of information from distributed feedback networks of the type shown in Figure 8 was devised by Kohonen in the early 1970's. Kohonen said that there exist at least two alternative adaptation laws according to which a genuine distributed associative memory can be formed. One of them is a biologically plausible principle called *Novelty Filter* in which the feedback is negative, tending to compensate for the input excitation. The second, with positive feedback, is of the Hopfield type, although the recalled information, in a selective form, is perfect only if the changes in the network are small, and the recalled information identifiable as a *perturbation component* in the signals. In both of these models, the feedback matrix, N, of Figure 8 is of the correlation matrix form $\cong E_k y^{(k)} y^{(k)T}$, or the sum of outer products of the output vectors.

Kohonen then discussed some four questions relating to different forms of memory:

1. Noise content and noise suppression in distributed associative memory. Kohonen had shown (Kohenen, 1984/1988) that the relative noise content of the recollection is roughly proportional to the inverse square root of the number of activated pattern elements in the key pattern. On the other hand, a statistically independent noise component in the key will be suppressed approximately by the same factor.

2. Damage resistance. Because information is represented in an ultimately redundant form in a distributed memory network, locally confined damage to the network node has a very graceful effect on recall accuracy. Kohonen has in fact represented the recall accuracy as a function of (randomly interconnected) nodes, from which calculation of degradation is directly discernible. The relative accuracy is roughly proportional to the inverse square root of the number of interconnections per node in this network.

3. Temporal patterns. The most salient capacity of biological memory relates to an ability to recall temporal sequences of patterns. To this end, the memory must be provided with delayed feedback, such that new patterns can be associated with their predecessors. Kohonen said that this mode of operation is obvious and was widely discussed in the early 1970's.

4. Synthesis of new recollections. Kohonen said that a widespread misconception concerns the stipulated selectivity of recall. It has been assumed that the memory should always selectively be able to decide upon and recall one of the memorized patterns, an aspect of which seems to underestimate all linear memory mappings. The "intelligent" memory, however, has to possess a certain ability of generalization, too; interpolation and extrapolation between memorized patterns is a very simple form of generalization, according to Kohonen. For instance, a linear memory, provided with a low degree of noise discrimination, is already able to solve an intelligence test task. In Figure 12, the uppermost row shows examples of 20 patterns which had the same macrostructure but different microstructures. The middle row shows three examples of "key patterns," which implies the associative recollections shown on the lowermost row. Although the microstructures of the keys were not stored in memory, the recollections in all the positions of the macrostructure were synthesized – not much unlike patterns that can be decomposed and resynthesized from the spatial frequency components. Kohonen said that one has to realize that neural networks may be able to perform syntheses of many different kinds of representations.

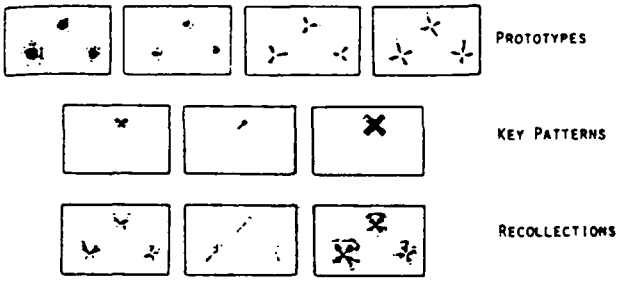


Figure 12. Demonstration of generalization in associative recall.

Parallel Architecture for Neurocomputers

P. Treleaven, who had previously presented a paper on programing languages for neurocomputers (above), surveyed current work on parallel neurocomputer architectures, concentrating on special-purpose hardware implementations and on general-purpose systems. He said that recent advances in neural computation models will only demonstrate their true value with the introduction of parallel computer architectures designed to optimize the computation of these models. There are three basic approaches for realizing neurocomputers: (1) special purpose neural network hardware implementations that are dedicated to specific models and therefore have potentially a very high performance; (2) neural network simulators using conventional hardware which are slow but allow implementation of a wide range of models, and (3) general-purpose neurocomputers that will provide a framework for executing neural models in much the same way that traditional computers address the problems of "number crunching," for which they are best suited. According to Treleaven, this framework must include a means of programing (i.e., operating system and programing languages), and the hardware must be reconfigurable in some manner.

Background. Even a small child can recognize faces, whereas a supercomputer is stretched to its limits performing such computations. In contrast, an inexpensive computer excels at a series of laborious calculations beyond most humans. This computational contrast between computers and humans is striking. Further it suggests, according to Treleaven, two fundamental domains of computation: *Symbol Processing* (computers) and *Pattern Processing* (humans).

In crude terms, the brain is a massively parallel natural computer composed of 10 to 100 billion brain cells (i.e., neurons), each neuron connected to about 10,000 others. Nuerons seemingly perform quite simple computations. The principle computation is believed to be the calculation of a weighted sum of its inputs, comparing this sum with a threshold, and forming its output if this threshold is exceeded. Yet the brain is capable of solving difficult problems of vision and language in about half a second (i.e., 500 milliseconds). This is particularly surprising given that the response time of a single neuron is in the millisecond range and taking into account propagation delays between neurons. Thus, it appears that the brain must complete these pattern-processing tasks in less than 100 steps.

The pattern-processing class of problems covering pattern recognition and learning applications are trivial for brains but are far from readily solvable by traditional (symbol processing) computers. Treleaven said that there has been a renewed belief that to solve demanding pattern-processing problems, parallel computing systems are needed which emulate the organization and function of neurons.

Since the basis of neurocomputers is consideration of the structure of brains, Treleaven briefly reviewed the key properties of neural systems as follows. The basic building block is the neuron. A neuron consists of a cell body called a soma, dendrites which receive input and branch out, and an axon that carries the output of the cells, one to another. Junctions between neurons, called synapses, occur either on the cell body or on spinelike extensions called dendrites. The neuron, in its simplest form, can be considered a threshold unit that collects signals at its synapses and sums them together using its internal summer. If the collected signal strength is great enough to exceed the threshold, a signal is sent out from the neuron by way of its axon.

Artificial Neural Networks

Artificial neural networks are neurally inspired mathematical models that use a large number of primitive processing elements (PE's) for pattern processing. Typically in neural networks, PE's are organized into layers, with each PE in one layer having a weighted connection to each PE in the next layer. This organization of PE's and weighted connections creates a neural network, also known as an artificial neural system (ANS). A neural network learns patterns by adjusting the strengths (weights) of the connections between PE's, analogous to synaptic weights. Through these adjustments a neural network exhibits properties of generalization and classification. Each component of the PE corresponds to a component of the neuron, as shown in Figure 13.

Neuron	Processing Element
Dendrites	Inputs
Synapses	Weights
Summer	Summation Function
Threshold	Threshold Function
Axon	Net Output

Figure 13. Correspondence of a neuron and PE.

Treleaven said that there are many different types of neural networks. Hecht-Nielson (1987) states that there are at least 30 different types of neural network models, currently being used in research and/or applications, of which 14 types are in common use. Perhaps the best known neural network models are the Hopfield model (Hopfield, 1982), the Boltzmann machine model (Ackley et al., 1985), and the Error Propagation model (Rumelhart and McClelland, 1986). Thus, neural networks are massively parallel interconnected networks of simple (usually) adaptive processing elements and their hierarchical organizations, which are intended to interact with the objects of the real world in the same way as biological nervous systems do.

Spectrum of Parallel Architectures

The set of possible parallel architectures for the basis of a neurocomputer, as shown by Figure 14, range from dedicated hardware (analogous in complexity to RAM's) to simulations on conventional computers, according to Treleaven.

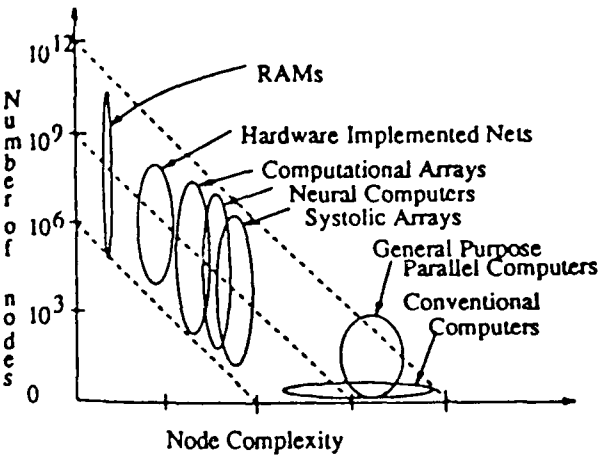


Figure 14. Spectrum of neurocomputer architectures.

There are three distinct approaches currently being taken for supporting neural network models:

- Special-Purpose Hardware – specialized neural network hardware implementations that are dedicated to a specific neural network model and therefore have a potentially high performance
- General-Purpose Neural Architectures – generalized neural computers for emulating a range of neural network models, thus providing a framework for executing neural models in much the same way that traditional computers address the problems of number crunching
- Simulations – neural network simulators using conventional hardware which are slow but allow support of a wide variety of models.

Research into parallel architectures for neurocomputers largely falls into two camps: special-purpose architectures and general-purpose architectures. Treleaven then examined the candidate neural network models for direct hardware implementation and also the candidate architectures for general-purpose neurocomputers.

Special-Purpose Architectures. Treleaven said that the approach for special-purpose neurocomputer architectures is to directly implement a specific neural network model in hardware to give a very high-performance system. Basically, any neural network model could be chosen, although currently a Hopfield associative memory model is typically favored because of its simplicity. Treleaven said that in general, neural network (or Connectionist) models are of two broad classes, namely, associative memories and categorization or learning systems.

With associative memories, information can be retrieved based on the content of the memory (autoassociator), or a relationship between remembered pieces of information (pair-associator). In addition, with both types of associated memory a corrupted "key" will lead to a recall of the nearest stored event.

With learning systems, data is presented repeatedly according to a set of rules, and the task is for the system to extract the underlying patterns. Learning systems can be further classified into supervised and unsupervised learning. During supervised learning, as in the Boltzmann machine and the "backpropagation of errors" algorithm, expected results govern the learning process. The "competitive learning" algorithm is an example of unsupervised learning.

Other neural network models include:

- Adaptive Resonance (ART) – a class of networks that form categories for the input data, and where the coarseness of the categories is determined by the value of a selectable parameter
- Backpropagation (BPN) – a multilayer network that minimizes mean square mapping error
- Bidirectional Associative Memory (BAM) – a class of single-stage heteroassociative networks
- Boltzmann Machine (BCM) – a class of networks that uses a noise process to find the global minimum of a cost function
- Brain State in a Box (BSB) – a single-stage autoassociative network that minimizes the mean square error
- Cerebellatron (CBT) – learns the averages of spatiotemporal command sequence patterns and relays these average command sequences on queue
- Counterpropagation (CPN) – a network that functions as a statistically optimal self-organizing lookup table and probability density function analyzer
- Hopfield (HOP) – a class of single-stage autoassociative networks without learning
- Lernmatrix (LRN) – a single-pass, nonrecursive, single-stage associative network
- Madaline (MDL) – a bank of trainable linear combiners that minimize mean square error
- Neocognition (NEO) – a multilayer hierarchial character recognition network
- Perceptron (PTR) – a bank of trainable linear discriminants
- Self-Organizing Map (SOM) – a network forming a continuous topological mapping from one compact manifold to another, with the mapping metric density varying directly with a given probablility density function on the second manifold.

According to Treleaven, two of the most important neural network models are the Hopfield model (i.e., autoassociator), which is typically chosen for special-purpose hardware implementation, and the backpropagation model, the most popular neural network learning model in use today. Treleaven said that the resurgence of interest in neural networks is largely due to Hopfield, who showed (Hopfield, 1982) that a neural network of interconnected processing elements will seek an energy minima. The Hopfield model acts on a binary input vector, I, mapping it to a binary output vector, O, both of n-elements, using a nxn weight matrix, W. The model comprises two alogorithms, namely, for storage and recall. The vector NET recalled from matrix is the same vector the neural network was taught, depending on the number of stored patterns. Although the Hopfield model is designed to store a single binary vector, it can easily be extended to store several binary vectors, according to Treleaven. He said that the Hopfield model is typical of a class of single-layer neural network systems, but many real-world problems cannot be represented by such neural networks. According to Treleaven, the solution to this problem is to introduce a third layer, called the hidden layer, between the input and output layers. The best known three-layer model is backpropagation.

General-Purpose Architectures. Treleaven said that the design of general-purpose parallel computers that are candidates for neurocomputer architectures, centers around a small set of parallel programing models. As shown in Figure 15, each programing model comprises a

COMPUTATION	DOMAIN					
Numeric	Processing		Symbolic	Processing		Pattern Processing
PROGRAMING	LANGUAGES					
Procedural	Object-Oriented	Single-Assignment	Applicative	Predicate Logic	Production System	Semantic Network
OCCAM, ADA	SMALLTALK	SISAL	Pure LISP	PROLOG	OPS5	NETL, IXL
COMPUTER	ARCHITECTURES					
Control Flow	Object-Oriented	Data Flow	Reduction	Logic	Rule-Based	Cellular Array
TRANSPUTER	DOOM	MANCHESTER	GRIP	ICOT PIM	NON-VON	CONNECTION MACHINE

Figure 15. Parallel computer architectures.

computer architecture and a corresponding category of programing languages. These parallel architectures and their associated programing languages have the following properties:

- Control Flow. In a control flow computer (for example, Sequent Balance, Intel iPSC, INMOS Transputer) explicit flows of control cause the execution of instructions. In their procedural languages (for example, ADA, OCCAM) the basic concepts are a global memory of cells assignment as the basic action, and explicit control for the execution of statements.
- Object-Oriented. In an object-oriented computer (for example, APIARY or DOOM) (see Recce and Treleaven, 1988) the arrival of a message for an instruction causes the instruction to execute. In an object-oriented language (for example, SMALLTALK or POOL) the basic concepts are: objects are viewed as active, they may contain state, and objects communicate by sending messages.
- Data Flow. In a data flow computer the availability of input operands triggers the execution of the instruction which consumes the inputs. In a single-assignment language (for example, SISAL, ID, LUCID) the basic concepts are: data "flows" from one statement to another, execution of statements is data driven, and identifiers obey the single-assignment rule.
- Functional. In a reduction computer (ALICE, for example, or GRIP) (Recce and Treleaven, 1988) the requirement for a result triggers the execution of the instruction that will generate the value. In an applicative language (such as Pure LISP, ML, or FP) the basic concepts are application of functions to structures, and all structures are expressions in the mathematical sense.
- Logic. In a logic computer (ICOT PIM or BULL DDC, for example) (Recce and Treleaven, 1988), an instruction is executed when it matches a target pattern and parallelism (or backtracking) is used to execute alternatives to the instruction. In a predicate logic language (for example, PROLOG) the basic concepts are that statements are relations of a restricted form, and that execution is a suitable, controlled logical deduction from the statements.
- Rule-Based. In a rule-based computer (NON-VON, for example, or DADO) (Treleaven, 1987), an instruction is executed when its conditions match the contents of the working memory. In a production system language (for example, OPS5) the basic concepts are: statements are IF...THEN...rules and they are repeatedly executed until none of the IF conditions are true.
- Cellular Array. In cellular array computers each processor is connected to its "near-neighbors" in a regular pattern that matches the flows of data and control in arrays (examples: Connection Machine, MPP, DAP, and CLIP) and systolic array processors (such as WARP). A class of programing languages corresponding closely to the computational arrays are semantic network languages such as NETL and IXL.

Treleaven said that when considering the above categories of parallel architectures as the basis of a neurocomputer, the most appropriate is cellular arrays. The other six categories are based on far more complex computational models than are required by the simple "threshold" models typical of neural computing. He said that the computational framework of cellular arrays is consistent with an idealized neural structure and supports well the distributed nature of data in these neural network models. An additional advantage is that frequently, even with current levels of VLSI processing, many processing elements can be fabricated on a single chip. Two specific cellular arrays, namely, the programmable systolic chip (Fisher, 1983) and the connection machine (Treleaven, 1987), are indicative of a "general-purpose" neurocomputer, according to Treleaven.

1. Programmable Systolic Chip. Programmable Systolic Chips (PSC's) can be assembled into a number of regular topologies (linear, two-dimensional array, etc.) to support the family of systolic algorithms. Once the PSC's are connected, they are configured for a specific systolic algorithm by down-loading identical code into each PSC. The PSC's then operate as a synchronous pipeline with the data being pumped from chip to adjacent chip. A PSC processor (Figure 16) consists of five functional units that operate in parallel and communicate simultaneously over the three buses. The five functional units are: a 64x60-bit microcode dynamic RAM and a microsequencer, a 64x9-bit DRAM register file, an ALU, and a multiplier-accumulator (MAC), plus three input and three output ports.

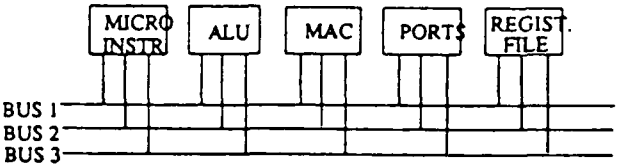


Figure 16. Programmable systolic chip.

2. Connection Machine. This machine is designed for concurrent operations on a knowledge base represented as a semantic network. A semantic network is a directed graph where the vertices represent objects (for example, sets) and the arcs represent binary relations (for example, set membership) required by the knowledge to be represented. A connection machine comprises 64K identical "intelligent" memory cells connected as a hypertorus structure. A connection machine (Figure 17) is a bit serial processor, comprising a few registers, an ALU, a message buffer, and a finite state machine. All cells are configured with the same program, known as the "rule table," which defines the next state and output functions of the finite machine. A cell reacts to an incoming mess-

age according to its internal state and the message type, and performs a sequence of steps that may involve arithmetic or storage operation on the contents of the message and the registers, sending new messages, and changing its internal state.

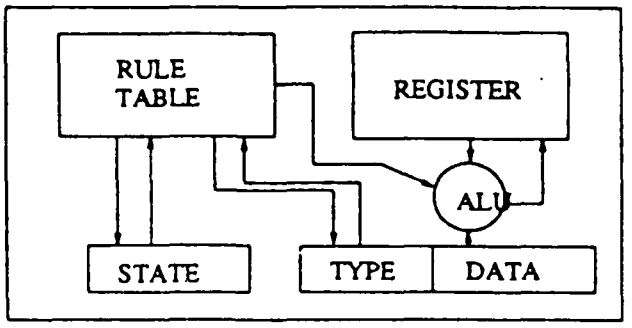


Figure 17. Connection machine "intelligent" memory cell.

Special-Purpose Neurocomputers

Treleaven said that when considering the implementation of neural network models, the basic corresponding hardware structure is the corssbar switch (Kohonen, 1987; Kuczewski, 1987), shown in Figure 18. This crossbar switch can be enhanced for neural network models by the introduction of lateral feedback. From this organization it is possible to devise a complete operational module for neural systems, as proposed by Kohonen (1987) and illustrated by Figure 18(b). Kohonen states that the most natural neural topology of a neural network would be two-dimensional and the distribution of lateral feedbacks within the system could be the same around every neuron.

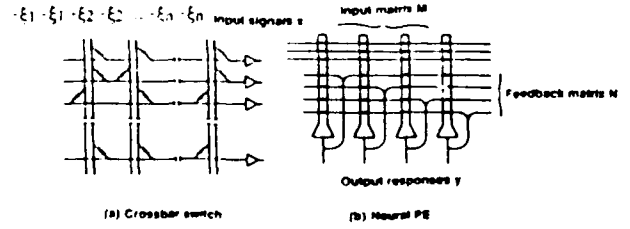


Figure 18. Neurocomputer components.

Treleaven said that developments of special-purpose, very-high-performance, typically analog circuits for neural networks are underway at a number of locations. Leaders in this field are: Jackel at AT&T Bell Laboratories (Holmdel), Lambe at the NASA Jet Propulsion Laboratories in Los Angeles, Mead at California Institute of Technology (Caltech), and Goser at the University of Dortmund, West Germany.

Implementing large numbers of individually primitive processing elements in VLSI technology is intuitively appealing. In addition, analog circuits generally occupy less area than the equivalent digital circuits. However, Treleaven added, a number of technology-dependent limitations are encountered. These are: (1) cost [chip area is the principal cost in VLSI], (2) power [analog processing elements typically require a high power consumption], and (3) parameter variation [fabrication of small devices introduces variations effecting the currents transferred].

Treleaven said that researchers at Caltech are investigating VLSI architectures for implementing neural networks, specifically networks based on the Hopfield model. The Caltech circuit for the Hopfield model is illustrated in Figure 19. This Hopfield circuit consists of three major components: amplifiers, interconnection matrix, and capacitors. The collection of amplifiers (cf. neurons) with gain function $V = g(v)$ are connected by the passive interconnection matrix which provides the unidirectional synapses, connecting the output of one neuron to the input of another. The strength of this interconnection is given by conductance Gif = Go Tij. Lastly, the capacitances determine the time evolution of the system.

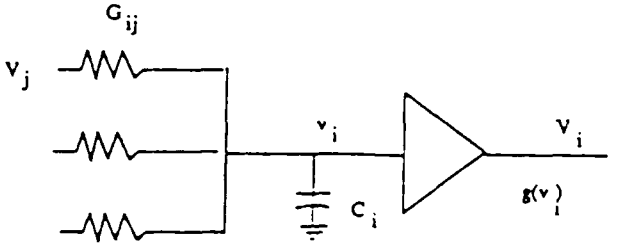


Figure 19. Circuit diagram for Hopfield model.

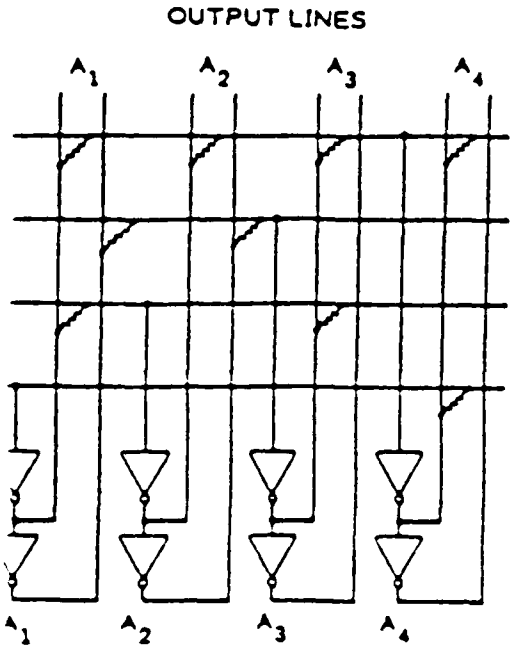
The Caltech chip, based on the above Hopfield circuit, contains 22 processing elements and a full interconnection matrix of 462 elements. The chip, fabricated in 4-μm NMOS technology, measures 6700 μm x 5700 μm, and has 53 I/O pads. This was followed by a 289-neuron CMOS chip.

AT&T are investigating CMOS associative memory chips that contain over 50 artificial neurons on a single chip using a combination of analog and digital VLSI technologies, together with a special microfabrication process. The chips are being used in pattern recognition where they perform feature extraction.

One associative memory chip implements a connectionist model of a neural network, and consists of 54 amplifiers plus a programmable coupling network where each amplifier can be connected to every other amplifier. Figure 20 shows a schematic of the implemented circuit. It consists of an array of 54 amplifiers with their inputs and outputs interconnected through a matrix of resistive coupling elements. All of these elements are programmable – i.e., a resistive connection can be turned on or off.

The connections between the individual "neurons" are provided by amorphous silicon resistors which are

ne CMOS chip in the last stage of fabrication on-beam direct-wiring. The associative mem- is fabricated in 2.3-μm CMOS and contains 00 transistors in an area 6.7x6.7 mm. Ninety he chip area is used for the coupling network. ests were made with 10 stable states of 40 bits gramed into the associative memory circuit. takes to converge to a stable state is between nanoseconds.



ire 20. AT&T associative memory schematic.

Purpose Neurocomputers

rding to Treleaven, neurocomputing is a fun- ly different domain of computation from tradi- nputing. Neurocomputing performs "pattern g," while traditonal computers perform "symbol g" specified by an explicit series of instructions. traditional computers are extremely flexible for rocessing. Treleaven said that what now is re- a complementary general-purpose neurocom- le to support a spectrum of neural network For a general-purpose neurocomputer, a num- operties are identifiable:

ular processing element. PE's should be modu- l hence replicatable, therefore each PE should elf-contained unit comprising processor, com- ations, and memory.

ive processing element. To make large neuro- uters (with millions of PE's) feasible, a PE must mitive and must allow a number to be packed on e VLSI chip or wafer.

ar communication. To allow neurocomputers to ensible, regular communications structures are required, especially to overcome connectivity limitations of VLSI.

- Asynchronous operation. With the potential to match the heterogeneous richness of the brain, neurocomputers may need to become multi-instruction multi-data stream (MIMD) devices.
- Programmability. For a neural computer to be general-purpose, and hence support a wide range of neural network models, the PE's must be programmable both in terms of interconnections and the function supported by a PE.
- Stability. Any asynchronous parallel system requires the processing and communications to provide inherent stability in all programed situations.
- Virtual processing elements. For a neurocomputer to execute potentially any massively parallel neural network, the concept of virtual processing elements that can be "paged" onto the neurocomputer from a backing store seems inevitable.

Treleaven then examined some of the general-purpose neurocomputers that have been developed, and said that neurocomputer development is a subject still in its infancy, hence the number of complete working neurocomputers is limited.

Treleaven said that in the US neurocomputing products are being marketed by such corporations as TRW, Hecht-Nielson Neurocomputer (HNC) (San Diego, California), Nestor Inc. (Rhode Island), Verac Inc. (San Diego, California), AIWARE Inc. (Cleveland, Ohio), Neural Systems Inc. (Vancouver, Canada), NCI (New Jersey), Neuraltech Inc. (Portila Valley), Neuronics Inc. (Chicago, Illinois) and SAIC (Tucson, Arizona). The pioneer of neurocomputer design is Hecht-Nielson, who has produced most of the commercially available general-purpose neurocomputers in which an arbitrary interconnectivity of the PE's can be defined. At TRW, Hecht-Nielson produced the Mark III and Mark IV machines. The Mark IV has 200,000 PE's, each capable of 25 interconnections. Subsequently, Hecht-Nielson's own company, HNC, has developed the ANZA system, a co-processor board which interfaces to an IBM PCAT. ANZA has 30,000 processing elements and allows a total of 300,000 interconnections between all the PE's. Lastly, Nester Inc. produces neurocomputer systems for hand-written-character-recognition based on the work of Cooper at Brown University, Providence, Rhode Island.

In Europe, neurocomputers have been produced by Alexsander of Imperial College, London, by Kohonen of Helsinki University, Finland, and by Garth of Texas Instruments (TI). Alexsander has developed a series of systems called Wisard. Wisard II is organized as a hierarchial network of PE's, with each PE being constructed from commercial RAM. The RAM's address inputs are used to detect binary patterns, with the input field for an image (comprising 512x512 binary pixels) being

connected to the first layer of PE's. Kohonen, a pioneer in associative "memory neural" networks, has experimented with several "neural network" distributed memories. He has recently completed a commercial-level neurocomputer based on signal process modules and working memories to define a set of virtual processing elements. This neurocomputer, optimized for speech recognition, allows 1000 virtual processing elements with 60 interconnnections. It can perform a complete spectral analysis and classification in phenomes every 10 ms. Lastly, Garth of TI, working with the University of Cambridge, UK, has developed NETSIM , a three-dimensional array of processing elements, each based on specially designed chips plus a 80188 microprocessor.

In Japan, H. Nakano of Tokyo University has completed a number of neurocomputers, some dating from 1970. The Association, his best known neural network system, is a hardware, correlation-matrix type of associative memory. A number of companies such as Japan's Fujitsu are also working on neurocomputers. The properties of the above neurocomputers are summarized in Figure 21.

Neurocomputer	virtual PEs	interconnects	updates/sec
HNC ANZA	30K	300K	25K
TRW MARK III	65K	1M	450K
TRW MARK IV	256K	5.5M	5M
IBM NEP	1M	4M	800K
NETSIM	256x27K	64Kx27K	4M

Figure 21. Comparison of Neurocomputers.

HNC ANZA. The ANZA Neurocomputer, developed and marketed by Hecht-Nielson Neurocomputer Corporation, is designed to support any neural network algorithm. The ANZA system comprises an ANZA neurocomputer coprocessor board for an IBM PC AT, a user interface subroutine library, and basic netware packages for the common neural network algorithms. The ANZA coprocessor board plugs into the backplate of a PC AT. The board is based on a Motorola M68020 plus a M68881 floating point coprocessor. With 4 Mbytes of dynamic RAM to store the network, ANZA is capable of implementing 30,000 PE's with 48,000 interconnections. These interconnections are updated at 25,000 interconnections per second during learning and 45,000 in feedforward mode.

The user interface subroutine library (UISL) is a collection of routines providing access to the ANZA system functions. Examples include: load network, set learning, etc. Lastly, the basic netware package contains five of the classic neural network algorithms in a parametized specification that can be configured for a specific user application. These algorithms are: Backpropagation, spaciotemporal (Formal Avalanche), neocognition, Hopfield (plus bidirectional associative memory) and counter-propagation networks. In these networks the interconnection geometry and the transfer equations are already specified. However, the number of PE's, their initial state and weight values, learning rates, and time constants are all user selectable.

TRW Mark III and IV. The TRW neurocomputer family consists of the Mark II simulator, the Mark III neurocomputer workstation, and the Mark IV high-speed neurocomputer (Kuczewski, 1987). All share the common artificial neural system environment (ANSE) user environment. The Mark III neurocomputer consists of up to 15 physical processors, each built from a Motorola M68020 microprocessor and a M68881 floating point coprocessor, all connected to a common VME bus. A neural network to be processed is distributed across the local memories of the PE's. Currently, the Mark III supports 65,000 virtual processing elements with over 1 million trainable interconnections, and can process 450,000 interconnections per second.

The Mark IV neurocomputer is a single high-speed, pipelined processor using virtual PE's and interconnection structure. Here the bulk of the hardware is devoted to forming the interconnections. The Mark IV supports 236,000 virtual processing elements with over 5.5 million trainable interconnections, and is capable of processing 5 million interconnections per second, including the preweighing function and learning law.

IBM NEP. IBM has developed a complete experimental neural network programing environment, called Computation Network Environment (CONE). CONE comprises: a network emulation processor (NEP – a cascadable parallel coprocessor for the PC); a network interactive execution program (IXP); and a high-level generalized network language (GNL). The major functional blocks of a NEP consists of six major units: a 5MIPS T1320 signal processor, a 64-K-word x 16-bit SRAM data memory, a 4-K-word x 16-bit SRAM program memory, and a 100 bytes/sec inter-NEP NEPBUS interface. Up to 256 NEP's can be cascaded in a unidirectional interprocessor communications network (NEPBUS), supporting in total 1 million virtual PE's and 4 million interconnections. To preserve interprocessor communication bandwidth, each NEP contains a high-speed "local I/O" unit for the attachment of real-time I/O devices. Both the NEPBUS and the local I/O interface are FIFO buffered, allowing a group of NEP's to asynchronously update the state of their respective portions of a large neural network. Each NEP can stimulate about 4,000 virtual PE's and 16,000 interconnections, with 30 to 50 complete network updates per second. The number of PE's emulated (by each NEP) can be increased by decreasing the total number of interconnections. In addition, the length of a network update cycle can be reduced by dividing a network across more of the NEP's, with a speed increase proportional to the number of NEP's.

IC Wisard. The Wisard systems are a series of neurocomputers specifically for image processing, developed by Alexsander, and commercialized as the Wisard/CRS1000 by Computer Recognition Systems Ltd. Design of a Wisard system centers on an array of RAM cells used as a set of discriminators for the image to be processed. These discriminators operate in an analogous way to a hologram. Consider the processing of a 512x512-bit binary image. From this binary image, groups of n bits are extracted to form n-tuples, using a random but fixed mapping. In this case the image contains 2**18 bits, so for n = 8, 2**15 n-tuples are taken. Each n-tuple is then used to address a specific RAM cell in a discriminator. Conceptually, in this example, 2**15 RAM cells, each of 256 bits are needed. However, for efficiency these cells can be grouped into discriminators. By using RAM arrays organized as k-bit words, k discriminators may be provided simultaneously.

Initially all cells in these discriminators are set to zero. During training, a discriminator is selected and 1's are entered into all cells addressed by the image. If this image is presented again, the effect is for the discriminator to produce a logical 1 on all its outputs. A partial image produces a reduced set of 1's.

CRS has developed the Wisard by adapting its CRS1000 automatic inspection system. It operates as follows: Initially a video image is captured, and Wisard fetches n-tuples from the image store, thresholding the data with a programmable comparator. The address in discriminator memory is made up from an autoincrementing counter and the 5-tuple (say ABCDE) read from the image memory. In Wisard up to 16 discriminators are directly supported by having a 16-bit wide memory discriminator memory. Tuples ranging from 4 to 8 tuples are sufficient for most industrial applications, according to Treleaven.

NETSIM. The NETSIM neurocomputer has been developed through a collaboration of Texas Instruments (UK) and Cambridge University (UK). A NETSIM system, as illustrated by Figure 22, consists of a collection of neural network simulator cards physically connected in a three-dimensional array, with a PC host acting as front-end. Each NETSIM card is an autonomous processing element, comprising an industry-standard 80188 microprocessor, two custom chips, a solution engine, and a communications processor, along with three memories for synapses, program, and BIOS.

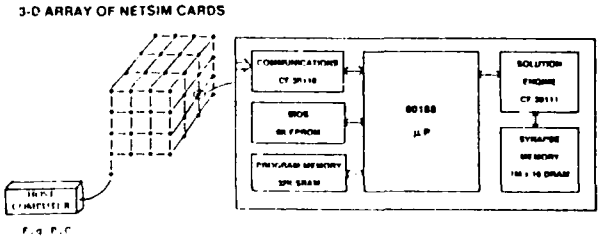


Figure 22. NETSIM neurocomputer system.

The solution enquire, CF30111, operates as a back-end vector processor for the microprocessor. Assuming a network "solution" is given by Oj = f(Ii.Tij), the solution enquire computes the sum of products between the input vector, I, and the relevant synapse vector, T, in its memory, and returns the result as a 16-bit integer to the microprocessor. The microprocessor then computes the nonlinear function, fO, to produce the output of the neuron. The output is then passed to the communications processor for transmission to the network to which the neuron is logically connected.

The solution chip performs four instructions: (1) *dummy cycle* – chip management [for example, clear register]; (2) *repeat-multiply-sum* – solve the network by multiplying an input vector by the corresponding synapse and adding it to the sum register; (3) *read-write* – move data within memory, allowing access to the synapse and input memory space from the microprocessor; and (4) *repeat-multiply-sum-write* – update synapses by multiplying two 8-bit vectors (for example, Input x Delta) with prescale, adding the product to a 16-bit term (Tij) and storing it as a new term (T'ij).

Organization of the communications chip centers on a 64-bit message register, where the first two bytes represent the message's destination address (relative to the sending mode) and the remaining bytes are for data. The addressing scheme allows messages to be transmittted to +/-15 NETSIM cards in each of three dimensions. According to Treleaven, the NETSIM neurocomputer has been shown to support the majority of common neural network algorithms including the Hopfield model and the Backpropagation model. Each NETSIM card is capable of solving rectangular networks at a rate of 4 million synapses per second and with learning at a rate of 1.3 million synaptic updates per second.

UCL Neuro-Chip. At University College, London, UK, Treleaven and coworkers designed and are currently implementing in CMOS a primitive processing element for building a parallel MIMD neurocomputer, configured from an array of these elements (Recce and Treleaven, 1988). The goal of the neurocomputer is to support a range of connectionist algorithms, spanning both neural network models and semantic network languages.

Each processing element, as shown in Figure 23, comprises three units: communication, processor, and local memory. The communication units, when interconnected by their bidirectional, point-to-point connections, support a logical bus structure for routing message packets. Each processing element has a neuron name, used for message routing. The processor consists of a primitive ALU, supporting ADD, SUB, AND, XOR, etc.;two visible registers, an instruction pointer IP, and an accumulator AX; and 16 instructions. All data, addresses, and instructions are 16 bits. Treleaven said that a processor with only two instructions could have been built, but this would have increased program sizes and hence the local

memory required. The memory is 4Kx16-bit words, although the instruction set allows a larger address space.

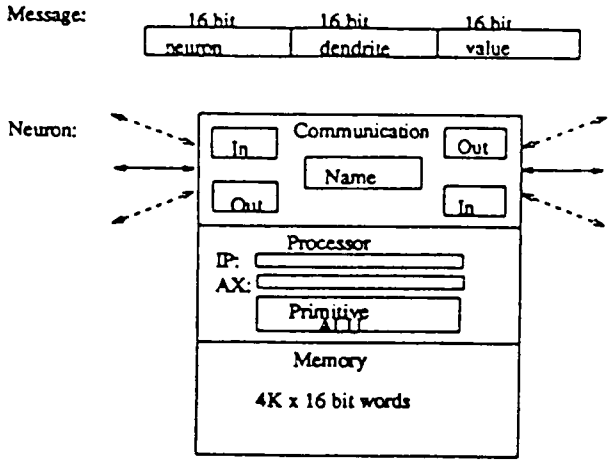


Figure 23. UCL MIMD "Neural" Processing Element.

This neurocomputer is configured by loading a simple program into each element; the code can be identical or different for each element. During operation, messages are sent from element to element. Each message (see Figure 23) consists of the neuron name (defining the destination processing element), the dendrite (defining the input link), and the value. When a message arrives at an element, an interrupt is generated and the message is processed by the neuronlike element. Treleaven said that this investigation of MIMD neurocomputers is still at an early stage, and he expects to design and fabricate a series of progressively simpler neural processing elements during the course of the project.

New Technologies

Treleaven said that advances in technology have always constituted a major driving force for computer development. Three technologies that could have a large impact on future computers are: in the short term, gallium arsenide (GaAs); in the medium term, optical devices; and in the long term, molecular devices. Research into new technolgies such as optical devices, has expanded rapidly in recent years. In addition, many researchers in these new technologies look to neurocomputing to provide parallel architecture to utilize these novel devices.

GaAs technology has made rapid progress in recent years, particularly in the area of digital chip complexity, according to Treleaven. When comparing GaAs with silicon, its two main advantages are higher switching speed and greater resistance to adverse environmental conditions. However, GaAs is inferior to silicon in terms of cost (of material and lower yield) and transistor count (related to yield and power consumption). For neurocomputers, packing density (i.e., miniaturization) of PE's would seem to be more important than switching speed. Thus, currently, GaAs does not seem to provide any major benefits compared to silicon for neurocomputers.

Optical techniques for information processing have made rapid advances in recent years. Within this area, the term "optical computing" is defined as: the use of optical systems to perform computations in one-dimensional or multidimensional data that are generally not images (*Applied Optics*, 1987). The goal of this work is to build an optical binary digital computer which uses photons as the primary information-carrying medium rather than electrons. The potential advantages of optical computers include:

- They generate high space-bandwidth and time-bandwidth products.
- They are inherently two-dimensional and parallel.
- Optical signals can propagate through each other in separate channels with essentially no interaction.
- Optical signals can interact on a subpicosecond timescale.
- Optical devices can, theoretically, be made orders of magnitude smaller than silicon devices.

Thus the potential of optical parallel computers for neurocomputers is clear, according to Treleaven.

In the longer term, molecular computers comprise an exciting research area, Treleaven said. Although no molecular computing device seems so far to have been constructed, the possibility of organic switching devices and conducting polymers may come about from current developments in polymer chemistry, biotechnology, the physics of computation, and computer science. Treleaven said that although there is no clear consensus as to the viability of molecular computing devices, the potential for collaboration with neurocomputer research is obvious.

Optical. Using optical computing devices, it is theoretically possible to fabricate integrated circuits that are smaller and faster than those based on electronic technology and have greatly increased density. Optical computing research is being pursued throughout the world. Major national projects include, in Japan, the $70 million, 6-year Optoelectronics Project; in Europe, the European Community's Joint Optical Bistability Project; and in the US, the Optical Circuitry Cooperative, centered at the University of Arizona, in Tucson. The largest single company commitment to optical computing is that of AT&T Bell Laboratories.

Optical computing research can be divided into: (1) optical digital computing [an optical supercomputer performing binary digital computations which use bistable or nonlinear optical devices to model electronic transistors] and (2) optical analog computing [an optical system performing pattern recognition computation which uses lenses for Fourier transform and convolution operations].

To build optical digital computers, the prerequisite is a three-port optical transistor that exhibits optical bis-

tability. Optical bistability is analogous to light-sensitive sunglasses: when you look at the sun they go dark (cf. off) and when you turn away they go light (cf. on). Candidate three-port optical transistors center on two technologies: a nonlinear Fabry-Perot interferometer and multiple quantum-well material.

With respect to optical analog computers, here the base technology is the spatial light modulator. A spatial light modulator, in general, modulates the light output as a function of the light intensity input. The device consists of the spatial light modulator, the detector, and three beams: write, readout and output. The amplitude (or phase) of the "readout" light beam is modulated as a function of the intensity of a controlling "write" beam, and the reflected product of this two-dimensional information pattern is the "output" beam.

The architecture of optical computers relates closely to the properties of optical technology. Optical computers are naturally parallel and can support global communications from arrays of transmitters to arrays of receivers at the speed of light. Parallelism in optical computing means the ability to perform a large number of operations simultaneously but independently, such as switching all the optical logic gates in an entire two-dimensional array.

Consider the three major functional units of a computer, namely, memory, processor, and input/output. In a classic von Neumann electronic computer the processor must access the memory (or input/output) sequentially. This form of execution is "single-instruction-single-data stream" (SISD). With an optical computer, in contrast, the three units can access each other simultaneously in parallel. This form of parallelism is "single-instruction-multiple-data stream" (SIMD), as supported by traditional array processors. As shown in Figure 24, an optical computer should be implementable as a large optical gate array with the three units being indistinguishable, and global communications being provided by a separate unit such as a computer-generated hologram.

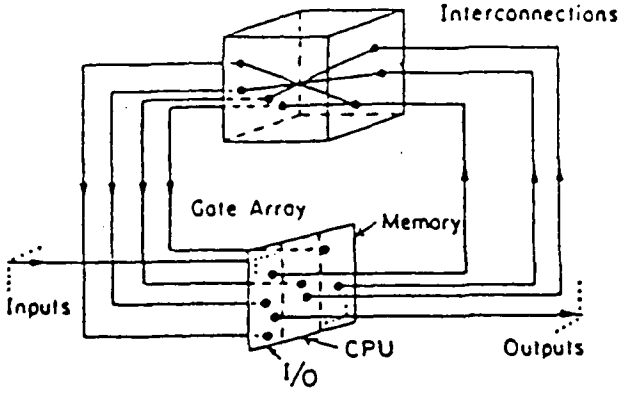


Figure 24. Possible optical computer architecture.

Molecular. Treleaven said that molecular computers are intended to be information processing systems made of proteins, and other large molecules, where these molecules sense, transform, and output signals. Molecular computing has its origins in the early 1970's when biological information processing models were first developed. Since that time, rapid progress has been made in biosensors, protein engineering, recombinant DNA technology, polymer chemistry, and artificial membranes. This research has culminated in major research programs, such as the Japanese government's 8-year, $65 million project under the auspices of the Research and Development Association for Future Electronic Devices.

The building blocks of a molecular computer are proteins and enzymes. A protein is a large molecule comprising smaller molecules, called amino acids, organized as a linear chain. A chain might typically contain 300 amino acids, chosen from among 20 commonly occurring types. An enzyme is a protein molecule that supports the pattern recognition. An enzyme is responsible for recognizing a "messenger" module (referred to as the substrate) and causing it to change to a "product" molecule. Each enzyme recognizes a specific type of messenger molecule by its geometric shape, and transforms it (i.e., switches its state) by making or breaking a precisely selected chemical bond.

The basis of this pattern recognition is the folding of protein chains. A protein assumes a shape when numerous weak interactions among the amino acids in the linear chain cause it to fold into an elaborate three-dimensional shape. Clearly, the number of different protein shapes is potentially enormous. In addition, the recognization process, involving the enzyme matching with the messenger molecule is itself sensitive to interaction with other molecules and local physicochemical conditions. This interaction is important, particularly for molecular computing, because an enzyme can also be switched into a different shape, thus allowing for memory and control at the molecular level. A molecular architecture might comprise three layers of molecules performing input, processing, and output. Input (i.e., receptor) molecules in one layer can transform the input signals into "messenger" (i.e., the substrate) molecules released inside the tactilizing medium. These sensory inputs might be light, temperature, or pressure. Processing molecules (i.e., tactilizing enzymes) interact with the messenger molecules, transforming them and causing a reaction-diffusion pattern of activity. Lastly, the output molecules (i.e., readout enzymes) read the local messenger molecules that result from the reaction, and generate the output signals from the computer.

Future Trends

Treleaven said that in the search for the "correct" parallel architecture for neurocomputers the desire for versatility (i.e., programmability) must be balanced against both the hardware complexity and the computational power. In Figure 14 (page 19), these trade-offs are presented pictorially. For neurocomputers the potential

complexity of PE's ranges from RAM cells to microcomputers like the INMOS company's Transputer. The near-neighbors of neurocomputers are the special-purpose hardware nets and cellular arrays.

To date, neural network models and applications have typically been developed through simulations on conventional computers such as DEC VAX. Due to severe performance constraints, these software simulations are being transferred to parallel computers such as the Intel iPSC Hypercube and INMOS Transputer-based systems. The recent developments of neurocomputer hardware components have been stimulated by these neural network software advances. These developments, as described above, subdivide into (1) special-purpose, analog implementations of specific models, typically the Hopfield model, and (2) more general-purpose neurocomputers, such as the HNC ANZA and the TI NETSIM.

With regard to the future, neurocomputers are believed to represent a fundamentally new pattern processing domain of computation, complementary to the symbol processing domain of traditional computers. This, Treleaven and others believe, recommends the development of a general-purpose parallel architecture for neurocomputers, whether analog or digital.

According to Treleaven, over the next 20 years neurocomputers can be expected to evolve through the following stages: (1) design of novel hardware components; (2) production of electrical neurocomputers; (3) design of components for optical neurocomputers; (4) production of hybrid electro-optical neurocomputers; (5) design of components for molecular neurocomputers; and (6) production of optical neurocomputers.

Combinatorial Optimization on a Boltzmann Machine

The problem of solving combinatorial optimization problems on a Boltzmann machine was discussed by J.H.M. Korst (Philips Research Laboratories, Eindhoven, the Netherlands). Korst showed that by choosing a specific connection pattern and appropriate connection strengths many combinatorial optimization problems could be mapped directly onto the structure of a Boltzmann machine. Thus maximization of the consensus in the Boltzmann machine is equivalent to finding an optimal solution of the corresponding optimization problem. The approach used by Korst and his collaborator at Philips (E.H.L. Aarts) was illustrated by Korst by the numerical results obtained by applying the model of Boltzmann machines to randomly generated instances of the max cut, the independent set and the graph coloring problem. From the results obtained by Korst and Aarts and coworkers, it was concluded that near-optimal solutions can be obtained by using in an efficient way the characteristic features of a Boltzmann machine, viz., massive parallelism and a distributed memory.

Korst said that recently many researchers have been turning their attention to the possibilities of neural networks and neural computing for carrying out complex computational tasks such as combinatorial optimization and learning. The characteristic features of neural networks are distributed memory (reduction of communications bottlenecks) and massive parallelism (fast computing). The model of the Boltzmann machine, introduced by Hinton and coworkers (Aarts and Korst, 1987) belongs to the class of neural network models and is a typical representative of connectionist models. The model of the Boltzmann machine was originally developed to carry out learning tasks within the field of pattern recognition, according to Korst. He said that the model exhibits interesting learning capabilities such as memory association and (restricted) induction. Futhermore, it provides a computational model that is especially suited for a massively parallel execution of the simulated annealing algorithm. As mentioned above, Korst and his group showed that the model can be used effectively within the field of combinatorial optimization, exploiting its massive parallelism.

A Boltzmann machine consists of a network of simple computing elements. The computing elements are considered as logic units having two discrete states, "on" or "off". The units are connected in some way. With each connection a connection strength is associated, representing a local quantitative measure for the desirability that the two connected units are both "on". A consensus function assigns to a configuration of the Boltzmann machine (i.e., a global state completely determined by the states of all individual units) a real number which is a quantitative measure that indicates to what extent the units have reached a consensus about their individual states, subject to the desirabilities expressed by the connection strengths. The state of an individual unit is iteratively adjusted by a stochastic function of the states of the units it is connected to and the associated connection strengths.

Korst presented details of the research carried out by him and coworkers on the feasibility of solving combinatorial optimization problems on a Boltzmann machine. However, I will not report the details at this time as a book by Korst and Aarts describes their work (Aarts and Korst, 1987). In summary, Korst said that the final solutions obtained by the Boltzmann machine algorithm are comparable to the final solutions of the simulated annealing algorithm. He said that for all three problems – the max cut, the independent set, and the graph coloring problem – the Boltzmann machine performs excellently; i.e., fast convergence to near-optimal results. The final solutions obtained by the Boltzmann machine are comparable to the solutions obtained by the simulated annealing problem. The precise choice of the connection strengths is not critical. The estimated speedup of the Boltzmann machine is linear with the number of units. As an overall conclusion, Korst stated that the Boltzmann machine can

exploit its massive parallelism and distributed representation efficiently in solving (or approximating) a wide range of combinatorial optimization problems. He said that this result becomes more significant when the Boltzmann machines are directly put on silicon, where each connection is hard-wired. In this way the annealing process can be performed extremely fast using analog devices which add up the incoming charge and perform the stochastic decision-making by using noise. The design of these hard-wired networks has been the subject of study for some time. Recently, Alspector and Allen (1988) presented a design of a VLSI chip with 5.10 gates, implementing a Boltzmann machine consisting of approximately 2000 units (this design is also suited for learning tasks). They estimate that their chip will run about a million times faster than simulations on a VAX. Korst said that optical implementations of the Boltzmann machine such as proposed by Ticknor and Barrett (1987) might even further increase this factor by some orders of magnitude. Korst said that his final conclusion is that the Boltzmann machines are promising as a means of solving (or approximating) combinatorial optimization problems. Clearly, however, more theoretical analysis and large-scale practical experience are needed to assess the real impact of the Boltzmann machine.

Neural Networks: A European Perspective

The European perspective was addressed by J.Y. Le Texier (Division of Electronic Systems, Thomson-CSF, Paris, France). He began his talk by saying that a formidable boom is now taking place in the US on the neurocomputing theme: federal agencies and large and small companies are launching their research programs, universities are setting up multidisciplinary specialized research centers and organizing special curriculae for students. At the same time, Japan talks about sixth-generation computers and the ambitious "Human Frontier Science Program." Le Texier then posed the question of how is Europe prepared to resist such competition in this emerging new technology. He believes that there is an extremely rich scientific potential in Europe for fundamental research in the field.

Le Texier said that the future of Europe in this domain is rather a matter of research coordination across disciplines and countries, and he presented a review of national and community funding programs currently encouraging such coordinated fundamental research efforts. If Europe is desirous, he said, of keeping up with international competition, it is also necessary to hasten technological transfers to industry; an ESPRIT-II proposal for a 5-year program gathering several major European companies was also presented.

The survey of European research resources was organized by Le Texier into three parts, each one around an area of research corresponding to a certain viewpoint in the field and a particular scientific background, as shown in Figure 25.

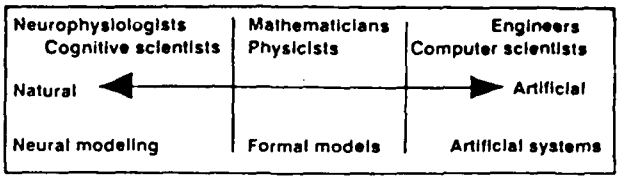


Figure 25. Poles of science and neural networks.

Neural Modeling

Neural modeling is the part of research investigating the structural and functional organization of animal brains from an experimental point of view, using anatomical, electrophysiological, and behavioral evidence. As an observational science it is an essentially bottoms-up approach, it is concerned with establishing bridges between high-level mental functions and neural activity. It often consists of generating models, explored through computer simulations, and incrementally modifying them to come to a better fit with observed data.

Le Texier said that numerous centers are carrying out such research work. To name only a few:

- R. Eckmiller, at the Department of Biocybernetics at the University of Dusseldorf, West Germany, whose research interests include the neural control of eye and hand movements in primates, as well as the development of neural networks for sensorimotor coordinate transform and motor program generation in intelligent robots
- W. Singer, at the Max Planck Institute for Brain Research, Frankfurt, West Germany, who is carrying out research on the mammalian visual system and collaborates with W. Phillips from the Department of Psychology and Computer Science, University of Stirling, UK
- A. Berthoz, at the CNRS laboratory of Sensory Neurobiology in Paris, France, whose activities include intracellular studies of neural circuits controlling eye movements and modeling of visual motion perception. (Berthoz and his group work in collaboration with A. Roucoux from the Laboratory of Neurophysiology, Catholic University of Louvain, Belgium)
- J.O. Keefe, University College, London, UK, doing research on the hyppocampus, the part of the brain supposed to be responsible for orientation capabilities in mammals
- D. Masterbroak, Biophysics Department, Groningen University, the Netherlands, who pursues an original line of research by studying the fly (*Calliphora erythrocephala*) and particularly, its visual system.

Formal Models

This part of Le Texier's presentation covered contributions of researchers with a background in theoretical sciences – mathematics, theoretical physics, theoretical computer science, etc. They use formal the-

oretical tools (statistical mechanics, automata theory, information theory, etc.) to study, in a top-down approach, the information processing capabilities of networks of elementary cells or processors.

The idea of modeling neurons by threshold automata can be traced back to work done by McCulloch and Pitts in 1942, according to Le Texier. A renewed interest in this area is due to the work in 1982 of the physicist, Hopfield, on associative memories, suggesting the use of networks of such automata to retrieve partially altered information. Le Texier said that the model had been derived by analogy with research on physical, disordered systems (spin-glass model).

A large number of theoretical physicists currently work on neural network models. Among them are:

- T. Kohonen (Finland) who has been working on associative memories and later set a mathematical framework on the theory of self-organization, based on neurophysiological evidence
- C. Von der Marisburg, who also works on models of organization of the brain, with application to the visual system (he currently works with E. Bienenstock on a model, radically original [according to Le Texier] providing an elegant solution to invariant pattern recognition)
- P.Peretto and J.J. Niez of the CEA-Nuclear Energy Commission, Grenoble, France, who have placed the description of associative memories in a probabilistic framework (Peretto also designed a hybrid analog-digital machine with impressive performance, Le Texier said)
- G. Dreyfus and L. Personnaz (ESPCI, Paris, France), who work on associative memories, on a Hopfield network, with particular learning algorithms (projection rule)
- D.J. Wallace (UK), E. Caianello (Italy), G. Toulouse (France), and J. Hertz (Denmark) – all theoretical physicists, who have had considerable influence on the development of connectionist research in their own countries.

In parallel with the physicists, mathematicians and theoretical computer scientists have been studying the dynamic properties and computational abilities of automata networks. Von Neumann initiated such research by looking for a model that would be valid for both living systems and machines: work on cellular automata and discrete iteration models is an offspring of these early studies. An important center of research is in Grenoble, France, at IMAG with contributions from F. and Y. Robert, M. Tchuente, J. Demongeot, and G.Y. Vichniac. Parallel processing capabilities of systolic arrays of an integrated processor, made possible by VLSI technology, renewed interest in the field, now intensively studied all over Europe.

Le Texier said that on another line of research – looking at supervised learning capabilities of automata networks – Y. Le Cun and F. Fogelman (University of Paris V, France) discovered the backpropagation algorithm (similar to Rummelhart's), now widely known, studied, and applied in experiments all over the world.

Artificial Neural Systems

This part of the research represents the effort to bring results in the preceding areas to the engineering realm. Such a task, requiring computer scientists' and engineers' contributions, expected to increase in the coming years, is two-fold: it must provide tools to support the research, particularly software simulation environments running on parallel architecture machines, and, on a longer term, offer solutions to the integration of the models on physical devices – it must validate the models, and verify their applicability by testing them on real-world applications.

Simulation of neural networks is extremely computer-intensive. One can either use supercomputers, general-purpose parallel hardware, dedicated massively parallel machines, or special-purpose integrated circuits.

Le Texier said that D.J. Wallace and his group at the University of Edinburgh, UK, in collaboration with D. Bounds at RSRE, Malvern, UK, make use of various hardware supports: the Meiko computing surface and the I.C.L. Distributed Array Processor. L.S. Smith at Stirling University, UK, also implements neural nets on Transputer-based machines. F. Roberts at IMAG uses an Intel hypercube and works on the design of a neurocomputer in collaboration with other research centers in Grenoble, including Herault's group at INPG, which has already developed a prototype machine for signal processing, according to Le Texier. At Texas Instruments, Bradford, UK, S. Garth also works on a low-cost, dedicated parallel machine, based on modular assembling of elementary boards. P. Treleaven, University College, London, works on the design of a massively parallel architecture.

Integration experiments are also being conducted: A. Murray, University of Edinburgh, UK, has designed a chip for the Hopfield model; M. Weinfeld, Ecole Polytechnique, Paris, France, implements the learning algorithm developed by G. Dreyfus and L. Personaz on a chip.

Among application studies, the most recent contributions include: R. Durbin (Oxford University, UK) and D.J. Willshaw (University of Edinburgh, UK), inspired by Von der Marlsburg's work, have developed the elastic net algorithm to solve the classical traveling salesman problem; J. Herault (Grenoble, France) has developed an algorithm performing signal separation on a mixed signal of independent source.

Funding of European Research

Up to now, according to Le Texier, European research has been funded by national scientific research in the various domains mentioned above. Le Texier then

sketched recent coordinated programs such as the West German government research program, the BRAIN initiative from the CEE, and a potential ESPRIT-II project for industrial research and development.

Germany. The government of West Germany has initiated major funding of approximately $6.1 million per year over a 10-year period starting in January, 1988. This concerted effort set up and funded by the Ministry of Research and Industry (BFMT) concerns eight research groups working on "Information Processing in Neural Architectures." Reinforcing the federal effort, the state of North Westphalia supports a program of research in neuro-informatics by establishing four endowed, tenured professorships, two at Düsseldorf University (strong background in neurophysiology) and two at Bochum (strong engineering background). The aim of these programs is to study the transfer of brain functions to computer science, with the specific goal of developing intelligent robots, working by federation of special-purpose integrated computers.

The BRAIN Initiative. As an initiative from DG XII of the CEE, a stimulation action, named BRAIN (Basic Research in Adaptive Intelligence), was launched in 1987. Its purpose is to support research collaboration aimed at a better understanding of how the brain works, and the design of machines capable of emulating some of its task-oriented problem-solving capacity.

A committee of experts has chosen six projects, for a level of funding close to 1 million ECU (about $1.2 million). The program is expected to show that with limited resource a great deal can be achieved by sharing expertise at a European level: in total, 28 labs and 100 researchers are involved. The list of accepted projects is as follows:

- Connectionist models for artificial intelligence (AI)
- Learning in automata networks: towards a neurocomputer
- Neural networks for data processing
- Distributed matrix memories
- Spatial and temporal transformations
- Graph-matching approaches of invariant perception.

An ESPRIT-II Proposal. The BRAIN research effort is a stimulation action, aimed at academic research centers. However, since the field is evolving towards a more industrial dimension, another level of community funding was necessary to keep Europe on a par with Japanese and American competition.

Although ESPRIT II is only a proposal, partners from eight different European countries involving large industrial companies (Thomson, Philips, Siemens) are now proposing to the CEE an ESPRIT-II project on neurocomputing. This 5-year project is conceived as an industrial R&D approach to connectionism, covering the spectrum of related technical issues. It is organized in distinct layers: applications, dedicated high-level language and simulation tools, parallel architecture support, and VLSI and WSI integration.

The idea is to demonstrate the utility of these techniques by developing a range of applications, mainly with application to the field of sensory-data processing (image processing, speech processing, robot control, etc.). They will provide the means for performance evaluation and refinements at all levels of the project.

As support tools for these studies, a connectionist software environment will be developed, including a high-level object-oriented dedicated language and a simulation and testing environment. These tools will be made available on existing European parallel computers and on workstations (Supernode, DOOM), establishing a de facto standard for European software interchange, which would be highly beneficial to the European research community, according to Le Texier. As a major undertaking, a massively parallel machine for emulation of connectionist models, a neurocomputer will be designed to provide the needed European hardware platform for the domain. In parallel, VLSI and WSI integration of specific models will be explored as a basis for future application-dedicated architectures.

According to Le Texier, integration of results stemming from neural network research is of critical importance for European information technology since those models appear to offer both a framework for taking advantage of hardware improvements, and a complement to symbolic Artificial Intelligence by providing real-time sensory-processing capabilities. In order to keep up Europe's level of competence in this emerging field, it is essential to organize European research, Le Texier said, and to encourage initiatives for creation of dedicated research centers, analogous to the Computer and Neural System Center at Caltech or the Center for Adaptive Systems at Boston University. Le Texier said that there should also be an increased communication between academic and industrial researchers in order to enhance coherence and coordination of the efforts and to prepare the industrialization of results.

Neurocomputing Applications: A United States Perspective

The US perspective was discussed by R. Hecht-Nielson (Hecht-Nielson Neurocomputer Corporation, San Diego, California). He first presented an overview of the subject, then said that since the invention of computing 45 years ago, there has been a strong desire to achieve information processing capabilities that are at least qualitatively similar to those possessed by animals. This desire is now beginning to be realized in the new field of neurocomputing. Neurocomputing is the engineering discipline concerned with nonprogramed, adaptive information processing systems called neural networks that develop transformation in response to the informa-

tion environment to which they are exposed. This new technology has been pioneered by scientists and engineers around the world and has an unusually international flavor. He said that neurocomputing would still not have emerged from the academic backwater in which it languished for 20 years were it not for key contributions by researchers in Europe and Japan.

Neurocomputing is a fundamentally new and different information processing paradigm. It is the first alternative to the programing paradigm that has dominated computing for the last 45 years, according to Hecht-Nielson. Neurocomputing and programed computing are fundamentally different approaches to information processing. Neurocomputing is based upon transformations, whereas programed computing is based on algorithms and procedures. What is being discovered, according to Hecht-Nielson, is that these two types of information processing, while conceptually incompatible, are highly complementary. He said that the emergence of neurocomputing as an intellectual pursuit has raised a number of fundamental questions. Among the most significant of these are the questions of how to define precisely what a neural network is, and how to assess the philosophical significance of neural networks. Answers to these questions are now beginning to emerge.

The Philosophical Significance of Neurocomputing

Hecht-Nielson said that neurocomputing is now recognized to be a new approach to information processing that differs at the most fundamental level from the programed computing paradigm that has been in use for the past 40 years. Beyond this identification of neurocomputing as being a fundamentally different information processing paradigm from programed computing, the question of the degree to which these paradigms differ is now becoming quite controversial. One school of thought, he said, holds that neurocomputing, while certainly based on adaptive principles, is at its heart, still based on the same arithmetic and logical foundations as ordinary, programed computing. This interpretation would suggest that the two paradigms are really not very different from one another. However, a second, more radical point of view has emerged within the last couple of years. This interpretation suggests that not only are the two computing paradigms totally different from one another in both conceptual and operational terms, but further, that the conceptual machinery developed to understand and explain algorithmic computing will be inadequate to understand the tricks that neural networks develop during training that allow them to carry out useful information-processing operations. For example, a neural network may be used to develop a highly accurate approximation to a mathematical function (the accuracy being measured externally in a least-means squared-error sense). Existing neural network theory may very well guarantee that the network will adaptively modify itself to achieve a high accuracy approximation to the function, and that this aspect of the network's operation will be fully understandable, according to Hecht-Nielson. However, on a deeper level, the network is actually using clever information-processing tricks internally (such as the development of features and internal representations that have particularly apt form, or the development of subtransformations within the neural network structure that have a particularly simple structure) that currently available information-processing concepts cannot be used for finding or understanding the operation of the neural network. The "San Diego Interpretation of Neurocomputing" holds that before such detailed understandings can be achieved, new theoretical tools specifically designed for the understanding of neurocomputing structures will have to be developed. The emerging debate between proponents of these two interpretations, according to Hecht-Nielson, is reminiscent of the heated intellectual battles triggered by the espousal of the Copenhagen interpretation of quantum mechanics 60 years ago.

The Technical State of Neurocomputing

Hecht-Nielson said that the recent upsurge in interest in neurocomputing has increased the rate of technical refinement of the subject. He said that great technical progress has been made within the last couple of years.

Neural Network Architecture. In neurocomputing the word "architecture" is reserved for the mathematical description of a neural network functional form. Architecture is an entirely separate issue from implementation – the physical realization of a neural network. Fundamentally new neural network architectures used to appear infrequently (at 2- to 3-year intervals), according to Hecht-Nielson. This pace has accelerated considerably. Having now gone through the process of inventing, developing, and promulgating more than 10 important neural network architectures, the neurocomputing community understands, he said, that this is a process that naturally takes a considerable amount of time. The three or four important new neural network architectures that have been discovered within the last year will require at least 1 to 2 years of additional development work before they are fit for application to real-world problems. He said that most neural network architects find it counterproductive to release their progeny before extensive theoretical and experimental exploration of the new network's properties can be carried out. This makes it highly advantageous for individuals and organizations within the neurocomputing application community to have close contact with the sources of architectural innovation. Given the startling increases in capability and performance that have been realized in recent neural network architectural advances, this close contact can make the difference between leading-edge capability and obsolescence. This is why, according to Hecht-Nielson, many neurocomputing companies have aligned themselves with one or more of the leading groups of neural network architects.

Neurocomputers. Hecht-Nielson said that it has now been thoroughly demonstrated that one of the major advantages of neurocomputing is the ability to automatically "parallelize" the implementation of neural network architectures in hardware. This can be accomplished both by the simplistic approach of implementing each individual processing element of a neural network in dedicated hardware, or by the more sophisticated approaches in which each item of connection hardware or each item of processing hardware is shared across multiple connections or processing elements. Such virtual neurocomputer designs can be far more cost effective than the fully implemented approach, according to Hecht-Nielson (although it is expected that for some future applications, the fully implemented approach is more appropriate).

In recent years, a large number of experimental neurocomputers have been built. Examples include large electronic neurocomputers such as the DARPA/TRW MARK IV (250,000 processing elements, 5 million connections per second during training), the Hughes Malibu Research Center all-optical neurocomputer (16,000 processing elements, 260 million connections-non-adaptive), and the Naval Research Laboratory electro-optic neurocomputer (65,000 processing elements, 4 billion connections). Neurocomputers on a chip have also been constructed, including the marvelous retina and choclea network chips of C. Mead at Caltech, according to Hecht-Nielson. Commercial general-purpose neurocomputers have also advanced significantly: from the world's first commercial offering (the TRW MARK III, introduced in April 1986; 8,000 processing elements, 400,000 connections, 350,000 connections per second with training; $70,000 list price) to Hecht-Nielson Corporation's newly announced ANZA Plus neurocomputer (1 million processing elements, 1 million connections, 1.5 million connections per second during training; $14,900 list price).

Neurosoftware Languages. Hecht-Nielson said that in programed computing, one of the most important fundamental innovations was the development of compilers and higher order programing languages that allow an algorithm to be readily expressed in a machine-executable form. Similarly, in neurocomputing, neurosoftware languages for describing arbitrary neural networks in a machine-implementable form have now been developed. Although the first-generation neurosoftware languages were only suitable for use by researchers, according to Hecht-Nielson, a second-generation language (HNC's AXON) that is specifically designed to support commercial application development as well as research, has now been developed. He said that AXON will be proposed as an industry standard.

The Neurocomputing Industry

Hecht-Nielson said that recent years have seen the introduction of commercial neurocomputing products by a number of start-up companies and a few established large corporations. This embryonic industry has the potential for large growth over the next few years. Established neurocomputing companies range from those providing a full spectrum of hardware, software, neurosoftware and customer support services to those positioned to exploit niche markets.

Hardware Products. Hecht-Nielson said that a small group of neurocomputing companies now offer hardware products specifically for use in implementing neural networks at high speed. These products range from board-level offerings from HNC and SAIC to the large TRW MARK III machine. This market is expected to expand rapidly and remain strong as these products and their derivatives begin to appear in end-use applications, according to Hecht-Nielson. Special-purpose neurocomputers designed for specific applications are also expected to appear within the next few years.

Software Products. Hecht-Nielson said that a number of companies have announced software products that allow existing computers to be used for implementing neural networks. Almost all of these products are oriented toward the educational market.

Neurosoftware Products. All computer manufacturers offer a collection of standard neural network architectures in neurosoftware form for the products. With the advent of AXON, these offerings will move from their current object code form to source code form, according to Hecht-Nielson. This will allow users to modify and customize neural network architectures for their particular applications.

Neurocomputing Applications. Hecht-Nielson said that application areas for neurocomputing seem to fall into four categories: pattern recognition, control, data analysis, and fuzzy knowledge processing. Except for small neural networks used for adaptive nulling of phase array radar antennas and adaptive equalization of telecommunications transmission lines, there are no neural network systems currently in everyday use. However, a large number of organizations within the industrial, commercial, and defense sectors are now pursuing significant applications. While most of these efforts have only recently begun, a few were started over a year ago, and some of them have now matured to the point where they are entering field testing. Some of these are expected to enter service within the next 1 to 2 years.

Commercial Applications. Commercial applications of neurocomputing will span an enormous range, according to Hecht-Nielson. They include loan application scoring, airline fare optimization, physical security systems, gas and petroleum exploration, movie special effects, and medical instrumentation. Other, more speculative applications, are also being developed, but these are not expected to mature within the next 2 years.

Industrial Applications. A number of significant industrial applications of neurocomputing are currently under active development, according to Hecht-Nielson.

These include manufacturing product inspection, process control, powerplant fault detection, computer-aided manufacturing, product warrantee history analysis, and computer security.

Military Applications. Military applications of neurocomputing that are currently being developed include radar and sonar target detection and recognition, multisensor fusion, steering and pointing control, image object detection and recognition, and application to electronic warfare, among others. The current study at the US Defense Advanced Research Projects Agency (DARPA) is expected to identify additional areas of potential payoff.

Early Indications of Success. According to Hecht-Nielson, between five and ten neurocomputing applications have now entered the field testing stage after demonstrating promising initial experimental results. Some of these may successfully cross the remaining institutional and operational barriers and become deployed applications within the next 1 to 2 years. Hecht-Nielson said that the fact that this many applications have gotten so far in such a short period of time is encouraging. It can be anticipated that even a small number of significant successes will lead to an explosive expansion in the volume of activity in neurocomputing.

Universities. As leaders in scientific and technological developments, many universities in the US have now started research and teaching programs in neurocomputing. Four universities (Boston University, Caltech, University of Southern California, and Stetson University of Florida) have developed formal education programs in neurocomputing. The University of California at San Diego offers neurocomputing courses in four different departments. Many other universities offer neurocomputing courses in at least one department. According to Hecht-Nielson, the volume of academic research in neurocomputing produced by American universities has increased markedly over the last 2 years. Almost all of the traditional journals in signal processing, image processing, pattern recognition, control, and information theory now regularly feature papers in neurocomputing.

Neurocomputing Industry. Hecht-Nielson said that new technologies always spawn new industries. Neurocomputing is no exception. Over 40 new neurocomputing start-up companies have been formed within the US during the last 18 months. The largest and most vertically integrated company in the industry is HNC (Hecht-Nielson's company), which offers a full range of neurocomputing hardware, software, and neurosoftware, as well as a complete range of services including training, contract research and development, original equipment manufacturer component development and production, and joint product development. HNC, a company with 50 employees located in San Diego, California and Washington, D.C., is the overwhelming sales leader in both the US and around the world. Hecht-Nielson said that HNC is particularly proud of its large international customer base in Japan and Europe. In Japan, HNC is represented by Sumisho Electronics, a subsidiary of the Sumitomo Corporation. Other leading US neurocomputing firms include SAIC (San Diego, California), TRW (San Diego, California), Nestor Inc. (Providence, Rhode Island), Neuronic (Cambridge, Massachusetts), and Synaptics (San Jose, California).These companies provide a wide variety of hardware, software, neurosoftware, and neural network chip technologies. Many other smaller companies such as BehavHeuristics (Silver Springs, Maryland) and Netrologic (San Diego, California) have been formed to pursue more specialized markets. For example, BehavHeuristics has developed a neural network-based system for optimizing airline fare decisions, and Netrologic offers neurocomputing applications R&D services.

US Government. Hecht-Nielson said that the renaissance of neurocomputing was at least partly triggered by the resumption of funding of research in this area by I. Skurnick of DARPA. This initial activity, beginning in the early 1980's, has now blossomed into a formidable ensemble of research and development activities being funded by practically every agency of the Department of Defense, as well as significant support for basic research in neurocomputing by the National Science Foundation. DARPA is currently conducting a major 6-month study (being managed by MIT Lincoln Laboratories) of the applications of neurocomputing to defense problems. This study is expected to yield recommendations for increased funding in a number of application areas. Hecht-Nielson said that all major US defense contractors now have neurocomputing activities underway. One such contractor is said to have over a hundred people working full time on neurocomputing applications research. Several compelling demonstrations of the efficacy of neurocomputing for solving defense problems in such areas as radar target classification, sonar target classification, object detection, and identification in images and electronic warfare have already been developed, according to Hecht-Nielson.

Conclusion

It is evident from the presentations at this conference that there is tremendous enthusiasm not only in the US and Japan but also in Europe in the area of neurocomputing. It is a technology that is not only intellectually stimulating but is also potentially valuable. The concensus at this meeting was that successful applications can be expected within the next 2 or 3 years. It also seems likely that the field will spawn a new academic discipline as well as new manufacturing and service industries. With neurocomputing now on the scene, the next few years may likely turn out to be a period of revolutionary expansion of our information processing capabilities.

References

Aarts, E.H.L. and J.H.M. Korst, "Computations in Massively Parallel Networks Based on the Boltzmann Machine," *Parallel Computing* (1987).

Ackerman, W.B., "Data Flow Languages," *IEEE Computer,* Vol. 15, No.2, (February 1982), 15-25.

Ackley, D.H., G.E. Hinton, and T.J. Sejnowski, "A Learning Algorithm for Boltzmann Machines," *Cognitive Science,* 9 (1985), 147-169.

Alspector, J. and R.B. Allen, "A Neuromorphic VLSI Learning System," *Advanced Research in VSLI: Proceedings of the 1987 Stanford Conference,* (Cambridge, Massachusetts: MIT Press, 1988).

Arvind, J. and K.P. Gostelow, "The U-Interpreter," *IEEE Computer,* Vol. 15, No.2, (February 1982), 42-49.

Brownston, L., *Programming Expert Systems in OPS5,* (Addison-Wesley, 1985).

Clark, K.L., and S. Gregory, "A Relational Language for Parallel Programming", *Proceedings of the ACM Conference on Functional Programming Languages and Computer Architecture,* (1981).

Dijkstra, E.W., "Guarded Commands, Nondeterminancy and Formal Derivation of Programs," *Communications of ACM,* 18 (1975), 453-457.

Fisher, A.L., "Architecture of the PSC: A Programmable Systolic Chip," *Proceedings of the Tenth International Symposium on Computer Architecture,* (June, 1983), 48-53.

Gardner-Medwin, A.R., "The Recall of Events Through the Learning of Associations Between their Parts," *Proceedings of the Royal Society,* London, B 194 (1976) 375-402.

Grossberg, S., *The Adaptive Brain II: Vision, Speech, Language, and Motor Control,* (Amsterdam: Elsevier/North Holland 1986).

Hanson, W.A., "CONE-Computational Network Environment," in *Proceedings of the IEEE First International Conference on Neural Networks,* (June 1978), 531-538.

Hebb, D.O., *The Organization of Behavior,* (New York: John Wiley, 1949).

Hecht-Nielson, R., "Neurocomputer Applications," in *Proceedings of the National Computer Conference,* (AFIPS, 1987) 239-244.

Higuchi, T. "A Semantic Network Language Machine," *Proceedings of the EUROMICRO Symposium,* ed. Waldschmidt, (North Holland, 1985), 95-104.

Hinton, G.E., T.J. Sejnowski, and D.H. Ackley, Boltzmann Machines: Constraint Satisfaction Machines that Learn, *Technical Report* CMU-CS-84-119, Carnegie-Mellon University (1984).

Hopfield, J., "Collective Computation with Continuous Variable," *Disordered Systems and Biological Organization* (1986).

Hopfield, J., " Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proceedings of the National Academy of Science* 79 (1982) 2254-2258.

Ingalls, D., "The Smalltalk-76 Programming System Design and Implementation", in *Proceedings of the Fifth ACM Symposium on Principles of Programming Languages,* (January, 1978), 9-15.

Kohonen, T., "State of the Art in Neural Computing", *Proceedings of the International Conference on Neural Networks, IEEE* (June 1987).

Kohonen, T. *Proceedings of the 2nd Scandinavian Conference on Image Analysis,* eds. E. Oja and O. Simula, (Helsinki: Suomen Hahmontunnistustutkimksen Seura, r.y., 1981), 214-220.

Kohonen T., *Proceedings of the Sixth International Conference on Pattern* (IEEE Computer Society, Siver Spring, MD, 1982), 114-128.

Kohonen, T., *Self-Organization and Associative Memory,* (Berlin-Heidelberg-New York-Tokyo: Springer-Verlag, 1984; Second ed., 1988).

Kovaly, K., *The Technology, the Players, the Potential,* (Fort Lee, New Jersey: Technical Insight Inc., 1987).

Kuczewski, R.M., "Neurocomputer Workstations and Processors: Approaches and Applications," *Proceedings of the First International Conference on Neural Networks* (June, 1987), 487-500.

McCulloch, W.S. and W. Pitts, "A Logical Calculus of the Ideas Immanent in Neural Nets," *Bulletin of Mathematical Biophysics* 5 (1943), 115-137.

Miller, R.K., "Implementing Associative Memory Models in Neurocomputers," *Neural Networks* (Madison, Georgia: SEAI Technical Publications, April 1987).

Minsky, M., and S. Papert, *Perceptrons,* (Cambridge, Massachusetts: MIT Press 1969).

Recce, M. and P. Treleaven, "Parallel Architectures for Neural Computers", ed. Eckmiller, Nato Workshop on Neural Computing, (Springer-Verlag, 1988).

Rosenblatt, F., *Principles of Neurodynamics* (New York: Spartan, 1962).

Rumelhart, D.E., G.E. Hinton, and R.J. Williams, "Learning Representations by Back-Propagating Errors", *Nature,* 323 (1986) 533-6.

Rumelhart, D.E., and J.L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition,* 1&2, (Cambridge, Massachusetts: MIT Press, 1986).

Rumelhart, D.E., and D. Zipser, "Feature Discovery by Competitive Learning", *Cognitive Science,* 9 (1985), 75-112.

Sejnowski,T., and C.R. Rosenberg, "NETTalk: A Parallel Network that Learns to Read Aloud," *Johns Hopkins University Technical Report* (JHIEECS-86101), 1986.

Shapiro, E.Y., "A Subset of Concurrent Prolog and its Interpreter," *Technical Report of ICOT* TR-003, February, (1983).

Shepherd, G.M., *The Synaptic Organization of the Brain,* (New York: Oxford University Press, 1974).

"Special Issue on Neural Networks", *Applied Optics Journal,* Vol. 26, No. 23, (December, 1987).

Ticknor, A.J. and H.H. Barret, "Optical Implementations in Boltzmann Machines," *Optical Engineering,* 26, 16 (1987)

Treleaven, P.C., "Computer Architectures for Artificial Intelligence," *Lecture Notes in Computer Science,* 272, (Springer-Verlag, 1987), 416-492.

Treleaven, P.C., and L. Gouveia, "Japan's Fifth Generation Computer Systems," *IEEE Computer,* Vol. 15, No. 8 (1982) 79-88.

Turner, D.A., "A New Implementation Technique for Applicative Languages," *Software-Practice and Experience,* 9, (1979) 31-49.

Ueda, K. "Guarded Horn Clauses", *ICOT Technical Report* TR-103, June, 1985.

Willshaw, D., "Holography, Associative Memory, and Inductive Generalization," *Parallel Models of Associative Memory,* ed. J.A. Anderson, (Hillsdale, New Jersey: Lawrence Erlbaun Associates, 1981).

Winston, P. and B. Horn, *LISP,* (Addison-Wesley, 1981).